Anexos Robot de acompañamiento emocional

Andres David Morales Baquero

Daniel Ruiz Martinez

Jose Daniel Juanias Martinez

Universidad Ean

Ingeniería

Ingeniería Mecatrónica

Bogotá, Colombia

03/04/2026

Contenido

Introducción **5**
Anexo A. Especificaciones generales y lista de materiales **7**
A.1 Especificaciones generales del sistema 7
A.2 Lista de materiales 7
Anexo B. Construcción física del prototipo **10**
B.1 Selección del peluche 10
B.2 Apertura y vaciado 10
B.3 Armazón interno del torso 11
Anexo C. Configuración del software base y de los modelos de inferencia **13**
C.1 Sistema operativo y configuración inicial 13
C.2 Motor de inferencia llama.cpp y modelo Gemma 2 2B 15
C.3 Reconocimiento y síntesis de voz 17
Anexo D. Subsistema electromecánico **19**
D.1 Conexión física del controlador PCA9685 19
D.2 Asignación de canales y límites de los servomotores 20
D.3 Controlador de servomotores 20
D.4 Calibración de los servomotores 22
D.5 Ojos con LEDs WS2812B 22
D.6 Biblioteca de expresiones 25
Anexo E. Subsistema conversacional **28**
E.1 Módulo de voz 28
E.2 Módulo de lenguaje 29
E.3 Memoria y aprendizaje del usuario 31
E.4 Mecanismo de seguridad ante situaciones de crisis 34
Anexo F. Integración y puesta en marcha **36**
F.1 Orquestador principal 36
F.2 Estructura de archivos del proyecto 39
F.3 Arranque automático 39
F.4 Optimizaciones de rendimiento 40
Anexo G. Verificación, cronograma y costos **42**
G.1 Lista de verificación previa a las pruebas con usuarios 42
G.2 Cronograma de implementación 43
G.3 Costo total estimado 43
Referencias bibliográficas **44**

Índice de tablas

Tabla A1: Especificaciones generales del prototipo Coco 7
Tabla A2: Lista de materiales del prototipo 8
Tabla B1: Criterios de selección del peluche 10
Tabla D1: Conexiones entre la Raspberry Pi 4 y el controlador PCA9685 19
Tabla D2: Mapa de canales del PCA9685 y límites de los nueve servomotores 20
Tabla D3: Conexiones del adaptador de nivel lógico y de los LEDs de los ojos 23
Tabla D4: Paleta de colores asociada a los estados emocionales de los ojos 24
Tabla E1: Datos que Coco aprende del usuario 32
Tabla F1: Pasos del ciclo de interacción del orquestador 36
Tabla F2: Optimizaciones de rendimiento y estabilidad 41
Tabla G1: Criterios de verificación de hardware y software 42
Tabla G2: Cronograma semanal de implementación 43

Índice de listados

Listado C1: Actualización del sistema y configuración inicial de la Raspberry Pi 14
Listado C2: Optimización de la memoria para la Raspberry Pi de 4 GB..... 14
Listado C3: Creación de la estructura del proyecto y del entorno virtual..... 15
Listado C4: Clonado y compilación de llama.cpp con soporte OpenBLAS..... 15
Listado C5: Descarga del modelo Gemma 2 2B Q4_K_M desde Hugging Face 16
Listado C6: Prueba de velocidad y calidad de la inferencia 16
Listado C7: Compilación de Whisper.cpp para el reconocimiento de voz 17
Listado C8: Instalación de Piper TTS y descarga de la voz española..... 17
Listado C9: Configuración del micrófono USB 18
Listado D1: Verificación de la detección del PCA9685 en el bus I2C..... 20
Listado D2: Controlador de servomotores (servo_driver.py) 21
Listado D3: Script de calibración de los servomotores (calibrar_servos.py)..... 22
Listado D4: Control de los LEDs de los ojos (ojos_led.py) 24
Listado D5: Biblioteca de gestos y expresiones del perrito (expresiones.py)..... 25
Listado E1: Módulo de captura, transcripción y síntesis de voz (voz.py) 28
Listado E2: Motor de lenguaje con historial de conversación y detección de emoción (cerebro.py)..... 30
Listado E3: Sistema de memoria vectorial y perfil del usuario (memoria.py) 32
Listado E4: Detección de crisis y recursos de ayuda (guardrails.py)..... 35
Listado F1: Orquestador principal del sistema (main.py)..... 37
Listado F2: Estructura de archivos y directorios del proyecto Coco 39
Listado F3: Script de inicio y servicio systemd para el arranque automático..... 40

Introducción

El presente documento reúne los anexos técnicos del proyecto de grado “Diseño e implementación de un dispositivo interactivo con inteligencia artificial para acompañamiento emocional”, en el cual se desarrolló el prototipo denominado Coco, un robot de acompañamiento emocional alojado en un peluche con forma de perrito. Su propósito es complementar el documento principal con el detalle constructivo y de implementación que, por su extensión y su carácter operativo, no resultaba apropiado incluir en el cuerpo del trabajo.

De este modo, se busca que cualquier persona con formación básica en electrónica y programación pueda comprender las decisiones de diseño adoptadas, reproducir el prototipo o auditar su funcionamiento. Para ello, cada anexo describe un conjunto coherente de elementos del sistema y presenta, de forma ordenada, los materiales, los procedimientos, las configuraciones y el código fuente empleados.

Conviene precisar que los anexos documentan la configuración local de referencia del sistema, en la cual la totalidad del procesamiento se ejecuta sobre una Raspberry Pi 4 de 4 GB de memoria: el reconocimiento de voz, la inferencia del modelo de lenguaje Gemma 2 de 2.000 millones de parámetros con cuantización Q4_K_M y la síntesis de voz, todo ello sin conexión a internet. Esta es la configuración que, en el capítulo de resultados del documento principal, se contrastó con la arquitectura híbrida (Tabla 2 de dicho documento).

El documento se organiza en siete anexos. El Anexo A presenta las especificaciones generales y la lista de materiales; el Anexo B detalla la construcción física del prototipo a partir del peluche; el Anexo C describe la configuración del software base y de los modelos de inferencia; el Anexo D documenta el subsistema electromecánico, conformado por los servomotores, los LEDs de los ojos y la biblioteca de expresiones; el Anexo E expone el subsistema conversacional, que comprende la voz, el modelo de lenguaje, la memoria del usuario y los mecanismos de seguridad; el Anexo F

explica la integración de los módulos y la puesta en marcha automática; y, finalmente, el Anexo G recoge la lista de verificación, el cronograma de implementación y el costo estimado.

Las tablas y los listados de código se identifican con la letra del anexo al que pertenecen seguida de un número consecutivo (por ejemplo, Tabla A1 y Listado C1). Los nombres de archivos, comandos y rutas se transcriben tal como deben introducirse en el sistema.

Anexo A. Especificaciones generales y lista de materiales

Este anexo resume las características generales del prototipo y enumera los componentes necesarios para su construcción, con el fin de que la adquisición de materiales pueda planificarse antes de iniciar el ensamblaje.

A.1 Especificaciones generales del sistema

La Tabla A1 sintetiza las características principales del prototipo Coco. Los valores de latencia y de costo corresponden a estimaciones de diseño; los resultados efectivamente medidos durante las pruebas se reportan en el capítulo de resultados del documento principal.

Tabla A1: *Especificaciones generales del prototipo Coco*

Característica	Especificación
Plataforma de cómputo	Raspberry Pi 4 Model B de 4 GB de RAM
Modelo de lenguaje	Gemma 2 2B, cuantización Q4_K_M
Actuadores	Nueve servomotores (cuello, boca, hombros, orejas y cola)
Indicadores visuales	LEDs RGB WS2812B ubicados en los ojos
Interacción por voz	Reconocimiento (Whisper) y síntesis (Piper) sin conexión a internet
Aprendizaje del usuario	Perfil estructurado y memoria semántica vectorial
Latencia de respuesta prevista en el diseño	2.5 a 3.5 segundos
Costo aproximado del hardware	140 a 190 €
Conectividad requerida en operación	Ninguna (funcionamiento 100 % offline)

A.2 Lista de materiales

Antes de iniciar el ensamblaje conviene reunir la totalidad de los componentes, ya que la compra por lotes reduce los costos de envío. Los servomotores y los módulos electrónicos pueden adquirirse en AliExpress, mientras que, para componentes críticos como el controlador PCA9685, se recomienda recurrir a proveedores como Adafruit o Amazon. La Tabla A2 detalla cada elemento con su modelo, cantidad, precio de referencia, fuente de adquisición y función dentro del sistema.

Tabla A2: *Lista de materiales del prototipo*

Componente	Modelo exacto	Cant.	Precio	Fuente	Función
Raspberry Pi 4	4 GB RAM — Model B	1	55 €	rpilocator.com	Cerebro del sistema
MicroSD A2 U3	Samsung 128 GB	1	16 €	Amazon	OS + modelos IA
Fuente RPi	USB-C 5V 3A oficial	1	10 €	Tienda RPi	Alimentar solo la RPi
Buck converter 5V/5A	LM2596 o MP1584	1	3 €	AliExpress	Fuente separada para servos
PCA9685	Módulo 16-ch PWM I2C	1	4 €	AliExpress	Controlador de 9 servos
Servo cuello Yaw	MG996R metal gear	1	4 €	AliExpress	Giro horizontal cabeza
Servo cuello Pitch	MG996R metal gear	1	4 €	AliExpress	Cabeza arriba/abajo
Servo cuello Roll	MG90S micro metal	1	2 €	AliExpress	Ladear cabeza
Servo boca	SG90 9g	1	1.5 €	AliExpress	Sincronización labial
Servo hombro izq.	MG90S micro metal	1	2 €	AliExpress	Brazo izquierdo
Servo hombro der.	MG90S micro metal	1	2 €	AliExpress	Brazo derecho
Servo oreja izq.	MG90S micro metal	1	2 €	AliExpress	Oreja izquierda
Servo oreja der.	MG90S micro metal	1	2 €	AliExpress	Oreja derecha
Servo cola	SG90 9g	1	1.5 €	AliExpress	Cola expresiva
LEDs WS2812B	Módulo circular 10mm ×2	2	1.5 €	AliExpress	Ojos RGB
Level shifter	3.3V→5V bidireccional	1	1 €	AliExpress	Para los LEDs WS2812B
Micrófono USB	MAONO o equiv. omnidirec.	1	15 €	Amazon	Captura voz STT
Altavoz + amp	3W 4Ω + PAM8403	1	4 €	AliExpress	Salida TTS
Capacitor electrolítico	470µF 16V	2	0.5 €	Tienda electrónica	Anti-ruido en servos
PowerBank	10000mAh USB-C	1	15 €	Amazon	Batería del sistema
Peluche de perrito	35–45 cm con ojos huecos	1	8 €	IKEA/Bazar	Carcasa blanda
Entretela termofusible	Pliego 50×100 cm	1	3 €	Mercería	Refuerzo interior tela
Plástico corrugado	Lámina 3mm 50×70 cm	1	2 €	Papelería	Armazón torso
Hilo de pesca	0.3mm y 0.5mm 10m	1	2 €	Ferretería	Transmisión de movimiento
Bridas nylon	Pack 3.6×150mm ×100	1	2 €	Bazar	Fijación de servos
Cables silicona	26 AWG, 5 m	1	3 €	AliExpress	Cableado flexible
Cables Dupont M-F	40 unidades 20cm	1	2 €	AliExpress	Conexiones GPIO
Cremallera invisible	15cm nylon	2	1 €	Mercería	Cierre de mantenimiento

Advertencia de seguridad. Los servomotores nunca deben alimentarse desde los pines de 5 V del puerto GPIO de la Raspberry Pi, pues los picos de corriente que generan pueden dañar la tarjeta de manera permanente. Por esta razón, el convertidor reductor (*buck converter*) se emplea como fuente dedicada para los servomotores.

Consideración sobre la memoria. El diseño del sistema se optimizó para la Raspberry Pi 4 de 4 GB de RAM. El modelo Gemma 2 2B Q4_K_M ocupa 1.5 GB, lo que deja 1.26 GB libres con todo el sistema en funcionamiento. Por este motivo se descartó el modelo Phi-3 Mini Q4_K_M (2.2 GB), cuyo tamaño no permite conservar el margen de memoria necesario.

Anexo B. Construcción física del prototipo

La carcasa del prototipo es un peluche comercial con forma de perrito, adaptado interiormente para alojar la electrónica y los mecanismos de movimiento. Este anexo detalla los criterios de selección del peluche, el procedimiento de apertura y vaciado, y la construcción del armazón interno que sostiene los componentes.

B.1 Selección del peluche

No cualquier peluche resulta adecuado para este fin. Se requiere un cuerpo con volumen suficiente para alojar la Raspberry Pi 4 (85 × 56 × 17 mm) junto con el PCA9685, la batería y el cableado, y una cabeza lo bastante grande para contener los tres servomotores del cuello. La Tabla B1 presenta los criterios de tamaño y construcción considerados.

Tabla B1: *Criterios de selección del peluche*

Característica	Mínimo	Recomendado	Justificación
Altura total	30 cm	35–45 cm	El torso necesita al menos 15×10×8 cm internos
Diámetro del torso	13 cm	16–20 cm	Alojar RPi4 + PCA9685 + batería en plano
Diámetro de la cabeza	8 cm	11–14 cm	3 servos apilados en el cuello + micrófono
Tipo de ojos	Cualquiera	Plástico hueco	Los LEDs WS2812B se colocan detrás — difunden luz
Tipo de relleno	Fibra cualquiera	Fibra hueca uniforme	Más fácil de retirar sin dejar grumos
Cola	Presente	Larga y flexible	Para montar servo de movimiento expresivo

B.2 Apertura y vaciado

El peluche posee una costura de cierre, normalmente ubicada en la espalda baja o en el vientre, que se convierte en la puerta de acceso permanente al interior. Abrirla con cuidado permite preservar la tela para el cierre definitivo. El procedimiento se realizó en las siguientes etapas:

1. Se fotografía el peluche completo desde todos los ángulos antes de intervenirlo, a fin de contar con una referencia para el cierre final.

2. Se localiza la costura de cierre (espalda baja o vientre) y se abre entre 12 y 15 cm con un descosedor, procurando no cortar la tela.
3. Se retira el relleno en el siguiente orden: torso completo, cuello completo, aproximadamente el 70 % de la cabeza (dejando algo de relleno en el hocico para conservar la forma), hombros y, si se instalará un servomotor en ella, la cola.
4. Se guarda el relleno en una bolsa, ya que puede emplearse posteriormente para rellenar las zonas que queden flácidas.
5. Se cortan parches de entretela termofusible para reforzar el cuello (dos piezas de 6 × 6 cm), los hombros (dos piezas de 5 × 5 cm), la zona lumbar (15 × 10 cm) y la base de la cola (5 × 5 cm). Se aplican con plancha, con la cara adhesiva hacia la tela interior, y se deja enfriar durante cinco minutos.

B.3 Armazón interno del torso

El armazón es una plataforma de plástico corrugado de 3 mm, cortada según la silueta interior del torso, que sostiene la Raspberry Pi 4, el PCA9685 y la batería mediante velcro desmontable. Su construcción se llevó a cabo de la siguiente manera:

1. Se corta la plataforma en plástico corrugado con dimensiones de 26 × 11 cm para un peluche de 35 cm (ajustando las medidas si el peluche es de mayor tamaño) y se redondean las esquinas con un radio de 15 mm.
2. Se perfora un agujero central superior de 16 mm de diámetro para el tubo de PVC de 8 mm que actúa como eje del cuello.
3. Se practican cuatro ranuras laterales de 2 × 40 mm para el paso de los cables de los servomotores hacia los brazos y la cola.
4. Se lijan todos los bordes cortados con lija de grano 120.
5. Se adhieren tiras de velcro (cara suave, 2 cm de ancho) en las posiciones destinadas a la Raspberry Pi 4 (zona media), al PCA9685 (zona media-derecha) y a la batería (zona baja).

6. Se pega espuma EVA de 5 mm en el perímetro de la plataforma para proteger la tela.
7. Se inserta verticalmente, por el agujero central, el tubo de PVC de 8 mm × 20 cm, que constituye el eje del cuello.
8. Se introduce la plataforma por la abertura del vientre y se cose a la entretela en seis puntos, con aguja curva e hilo encerado.
9. Se sella el tubo de PVC a la plataforma con tres vueltas de hilo y un punto de termofusible.

Anexo C. Configuración del software base y de los modelos de inferencia

Este anexo documenta la preparación del sistema operativo de la Raspberry Pi 4, la compilación del motor de inferencia, la instalación del modelo de lenguaje y la puesta en funcionamiento del subsistema de voz. Todos los comandos se ejecutan en la Raspberry Pi mediante una conexión remota por SSH.

C.1 Sistema operativo y configuración inicial

Se utilizó Raspberry Pi OS Bookworm de 64 bits en su versión Lite, es decir, sin entorno de escritorio, con el fin de maximizar la memoria disponible: el escritorio consume aproximadamente 200 MB adicionales que el modelo de lenguaje necesita. La preparación de la tarjeta microSD se realizó así:

1. Se descarga Raspberry Pi Imager desde https://www.raspberrypi.com/software/.
2. Se selecciona Raspberry Pi OS Lite (64-bit), basado en Bookworm.
3. Antes de escribir la imagen, se abren las opciones avanzadas (icono de engranaje) y se configura lo siguiente:
    a) Habilitar SSH con autenticación por contraseña.
    b) Establecer el nombre de host coco.local.
    c) Crear el usuario pi con la contraseña elegida.
    d) Configurar la red WiFi con su nombre y contraseña.
4. Se escribe la imagen en la microSD de 128 GB y se espera a que el proceso finalice.
5. Se inserta la tarjeta en la Raspberry Pi 4, se conecta la alimentación y se aguardan 90 segundos para el primer arranque.
6. Se establece la conexión remota con el comando `ssh pi@coco.local`.

Una vez operativo el sistema, se actualizan los paquetes, se instalan las herramientas de desarrollo y las bibliotecas de audio, y se habilita la interfaz I2C requerida por el controlador PCA9685 (Listado C1). Tras el reinicio, se restablece la conexión SSH y se aplican las

optimizaciones de memoria para el equipo de 4 GB (Listado C2): reducción de la memoria asignada a la GPU, sobrerreloj moderado de la CPU, desactivación de Bluetooth y desactivación del espacio de intercambio (*swap*), dado que la velocidad de la microSD lo vuelve ineficaz.

Listado C1: *Actualización del sistema y configuración inicial de la Raspberry Pi*

```
# Actualizar todo el sistema
sudo apt update && sudo apt full-upgrade -y

# Paquetes de desarrollo esenciales
sudo apt install -y git cmake build-essential python3-venv python3-dev
sudo apt install -y libopenblas-dev i2c-tools portaudio19-dev
sudo apt install -y libsndfile1 ffmpeg libasound2-dev sox

# Habilitar I2C para el PCA9685
sudo raspi-config nonint do_i2c 0

# Habilitar audio de 32 bits para mejor calidad TTS
sudo raspi-config nonint do_audio 1

# Verificar I2C activo
ls /dev/i2c*   # debe mostrar /dev/i2c-1

# Reiniciar
sudo reboot
```

Listado C2: *Optimización de la memoria para la Raspberry Pi de 4 GB*

```
# Editar configuración del firmware
sudo nano /boot/firmware/config.txt

# Añadir/modificar estas líneas al final:
# ── Minimizar memoria de GPU (solo necesitamos CPU) ──
gpu_mem=16

# ── Overclock moderado: +15% rendimiento sin riesgo ──
arm_freq=1800
over_voltage=2

# ── Deshabilitar Bluetooth (libera recursos) ──
dtoverlay=disable-bt

# Guardar con Ctrl+O, salir con Ctrl+X

# Desactivar swap (la microSD es demasiado lenta para ser útil)
sudo systemctl disable dphys-swapfile
sudo dphys-swapfile swapoff
sudo swapoff -a

# Verificar RAM libre disponible
free -h   # debe mostrar ~3.6 GB total disponible

sudo reboot
```

Finalmente, se crea la estructura de directorios del proyecto y el entorno virtual de Python que aísla las dependencias del sistema (Listado C3).

Listado C3: *Creación de la estructura del proyecto y del entorno virtual*

```
# Crear directorio del proyecto
mkdir -p ~/coco/{logs,scripts}
mkdir -p ~/models
mkdir -p ~/voices
mkdir -p ~/coco_memoria

# Crear entorno virtual Python 3.11
python3 -m venv ~/coco_env
source ~/coco_env/bin/activate

# Dependencias hardware
pip install adafruit-circuitpython-pca9685
pip install adafruit-circuitpython-motor
pip install rpi-ws281x adafruit-circuitpython-neopixel
pip install RPi.GPIO

# Dependencias audio
pip install sounddevice soundfile pyaudio numpy scipy

# TTS
pip install piper-tts

# Memoria y embeddings
pip install chromadb
pip install sentence-transformers

# Utilidades
pip install requests psutil

# Verificar instalación
python3 -c "import adafruit_pca9685; print('PCA9685 OK')"
python3 -c "import neopixel; print('NeoPixel OK')"
python3 -c "import sounddevice; print('Audio OK')"

echo "Entorno Python listo"
```

C.2 Motor de inferencia llama.cpp y modelo Gemma 2 2B

El motor de inferencia llama.cpp permite ejecutar el modelo Gemma 2 2B (Gemma Team, 2024) directamente sobre la CPU de la Raspberry Pi 4. La compilación con soporte para OpenBLAS y con las banderas de optimización propias de la arquitectura ARM Cortex-A72 es esencial: sin ellas, la velocidad de inferencia se reduce aproximadamente a la mitad. El proceso de compilación toma cerca de 15 minutos en este equipo (Listado C4).

Listado C4: *Clonado y compilación de llama.cpp con soporte OpenBLAS*

```
# Clonar el repositorio
git clone https://github.com/ggerganov/llama.cpp ~/llama.cpp
cd ~/llama.cpp

# Compilar con OpenBLAS y flags ARM Cortex-A72 optimizados
cmake -B build \
  -DGGML_BLAS=ON \
  -DGGML_BLAS_VENDOR=OpenBLAS \
  -DCMAKE_BUILD_TYPE=Release \
  -DCMAKE_C_FLAGS="-march=armv8-a+fp+simd -mtune=cortex-a72" \
  -DCMAKE_CXX_FLAGS="-march=armv8-a+fp+simd -mtune=cortex-a72"

# Compilar usando los 4 cores (tarda ~15 min, es normal)
```

```
cmake --build build --config Release -j4

# Verificar que compiló correctamente
~/llama.cpp/build/bin/llama-cli --version
# Debe mostrar: version: XXXX (commit hash)
```

El modelo seleccionado fue Gemma 2 2B en su variante Q4_K_M, una cuantización de 4 bits de la familia K-quants en su versión media. Esta variante ocupa 1.5 GB, por lo que cabe con margen en los 4 GB de memoria del equipo, y alcanza entre 38 y 48 tokens por segundo, valor consistente con el reportado en la Tabla 2 del documento principal para la configuración local. Su descarga se realiza desde Hugging Face (Listado C5) y, antes de continuar, se verifica la velocidad y la calidad de la inferencia mediante una prueba de generación (Listado C6).

Listado C5: *Descarga del modelo Gemma 2 2B Q4_K_M desde Hugging Face*

```
# Instalar cliente de Hugging Face
pip install huggingface-hub

# Autenticarte (necesitas cuenta en huggingface.co)
# Primero acepta la licencia de Gemma en:
# https://huggingface.co/google/gemma-2-2b-it-GGUF
huggingface-cli login
# Pegar tu token de acceso cuando lo pida

# Descargar el modelo cuantizado (1.5 GB, tarda 10-20 min)
huggingface-cli download \
  google/gemma-2-2b-it-GGUF \
  gemma-2-2b-it-Q4_K_M.gguf \
  --local-dir ~/models/

# Verificar descarga
ls -lh ~/models/gemma-2-2b-it-Q4_K_M.gguf
# Debe mostrar: ~1.5G
```

Listado C6: *Prueba de velocidad y calidad de la inferencia*

```
# Test de velocidad (benchmark)
~/llama.cpp/build/bin/llama-bench \
  -m ~/models/gemma-2-2b-it-Q4_K_M.gguf \
  -t 4 --ctx-size 512 -n 64
# Resultado esperado: 38-48 tok/s en eval

# Test de conversación empática
~/llama.cpp/build/bin/llama-cli \
  -m ~/models/gemma-2-2b-it-Q4_K_M.gguf \
  --prompt "<start_of_turn>user
Me siento muy triste hoy, no quiero hacer nada.<end_of_turn>
<start_of_turn>model
" \
  -n 80 -t 4 --ctx-size 1536 --temp 0.72 \
  --no-display-prompt
# Debe responder con empatía en 2-3 segundos
```

C.3 Reconocimiento y síntesis de voz

El subsistema de voz se compone de dos herramientas que funcionan sin conexión a internet. Para el reconocimiento de voz se compiló Whisper.cpp (Radford et al., 2022) con el modelo base, que ocupa 140 MB, presenta una latencia aproximada de 600 ms y ofrece una calidad suficiente en español (Listado C7). Para la síntesis se instaló Piper TTS con una voz española de alta calidad y una latencia aproximada de 250 ms (Listado C8). Por último, se configuró el micrófono USB, ajustando el dispositivo de captura y su nivel de entrada (Listado C9).

Listado C7: *Compilación de Whisper.cpp para el reconocimiento de voz*

```
git clone https://github.com/ggerganov/whisper.cpp ~/whisper.cpp
cd ~/whisper.cpp

cmake -B build \
  -DWHISPER_BLAS=ON \
  -DWHISPER_BLAS_VENDOR=OpenBLAS \
  -DCMAKE_C_FLAGS="-march=armv8-a+fp+simd" \
  -DCMAKE_CXX_FLAGS="-march=armv8-a+fp+simd"
cmake --build build --config Release -j4

# Descargar modelo base en español (140 MB)
bash ./models/download-ggml-model.sh base

# Probar con grabación de 5 segundos
arecord -d 5 -f S16_LE -r 16000 -c 1 /tmp/test.wav
~/whisper.cpp/build/bin/whisper-cli \
  -m ~/whisper.cpp/models/ggml-base.bin \
  -l es -f /tmp/test.wav --no-timestamps -t 4
# Debe transcribir lo que dijiste en ~600ms
```

Listado C8: *Instalación de Piper TTS y descarga de la voz española*

```
source ~/coco_env/bin/activate
pip install piper-tts

# Descargar voz española de alta calidad
mkdir -p ~/voices
cd ~/voices

# Voz davefx-medium: la mejor calidad/velocidad para RPi 4
wget -q "https://huggingface.co/rhasspy/piper-
voices/resolve/main/es/es_ES/davefx/medium/es_ES-davefx-medium.onnx" \
  -O es_ES-davefx-medium.onnx

wget -q "https://huggingface.co/rhasspy/piper-
voices/resolve/main/es/es_ES/davefx/medium/es_ES-davefx-medium.onnx.json" \
  -O es_ES-davefx-medium.onnx.json

# Probar síntesis (debe generar audio en ~250ms y reproducirlo)
echo "Hola, soy Coco. Estoy aquí para escucharte." | \
  piper --model ~/voices/es_ES-davefx-medium.onnx \
  --output_file /tmp/test_tts.wav && aplay /tmp/test_tts.wav

ls -lh ~/voices/
# Debe mostrar: ~63MB para el .onnx
```

Listado C9: *Configuración del micrófono USB*

```
# Listar dispositivos de audio disponibles
arecord -l
# Busca tu micrófono USB en la lista, anota el número de tarjeta

# Configurar micrófono USB como dispositivo por defecto
# Reemplaza 'card X' con el número de tu micrófono
cat > ~/.asoundrc << 'EOF'
pcm.!default {
    type asym
    playback.pcm {
        type plug
        slave.pcm "hw:0,0"
    }
    capture.pcm {
        type plug
        slave.pcm "hw:1,0"
    }
}
EOF
# Nota: hw:1,0 = tarjeta 1 (USB mic), ajusta si es diferente

# Ajustar nivel del micrófono al 80%
amixer -c 1 set Capture 80%

# Probar captura + transcripción completa
arecord -d 4 -f S16_LE -r 16000 -c 1 /tmp/mic_test.wav && \
~/whisper.cpp/build/bin/whisper-cli \
  -m ~/whisper.cpp/models/ggml-base.bin \
  -l es -f /tmp/mic_test.wav --no-timestamps -t 4
```

Anexo D. Subsistema electromecánico

Este anexo documenta los elementos que otorgan expresividad física al prototipo: el controlador PWM y los nueve servomotores que articulan la cabeza, la boca, los hombros, las orejas y la cola, los LEDs que conforman los ojos y la biblioteca de expresiones que coordina todos estos elementos con cada estado emocional.

D.1 Conexión física del controlador PCA9685

El controlador PWM PCA9685 se comunica con la Raspberry Pi mediante el bus I2C y gestiona los nueve servomotores del prototipo. En la Tabla D1 se detallan las conexiones de alimentación lógica, de datos y de potencia.

Tabla D1: *Conexiones entre la Raspberry Pi 4 y el controlador PCA9685*

Pin RPi 4	Nombre	Pin PCA9685	Cable	Función
Pin 1	3.3V	VCC	Rojo fino	Alimentación lógica del PCA9685
Pin 3	GPIO2 / SDA1	SDA	Azul	Línea datos I2C
Pin 5	GPIO3 / SCL1	SCL	Amarillo	Reloj I2C
Pin 6	GND	GND	Negro fino	Tierra lógica
Buck V+ (5V)	Fuente servos	V+ terminal	Rojo grueso	Potencia para los servos
Buck GND	Fuente servos	Tierra unificada con la RPi	Negro grueso	Tierra unificada con la Raspberry Pi

Resulta indispensable unificar las tierras del sistema. El cable negro grueso del convertidor reductor (GND de la fuente de los servomotores) debe conectarse al GND del PCA9685, en el terminal de potencia, y también al GND de la Raspberry Pi (pin 6 o cualquier otro pin GND). Sin una tierra común, los servomotores presentan movimientos erráticos. Una vez realizado el cableado, se verifica que el módulo sea detectado en el bus I2C (Listado D1).

Listado D1: *Verificación de la detección del PCA9685 en el bus I2C*

```
# Debe mostrar '40' en la dirección 0x40
i2cdetect -y 1

#     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
# 40: 40 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

D.2 Asignación de canales y límites de los servomotores

Cada canal del PCA9685 se asignó a un servomotor específico, al cual se le definieron un rango angular de seguridad y una posición neutra, como se muestra en la Tabla D2. Estos límites evitan que los mecanismos fuercen la estructura del peluche o se traben con la tela.

Tabla D2: *Mapa de canales del PCA9685 y límites de los nueve servomotores*

Canal	Servo	Modelo	Rango	Neutro	Ubicación física
CH 0	Cabeza Yaw	MG996R	50°–130°	90°	Cuello — giro izq/der
CH 1	Cabeza Pitch	MG996R	60°–120°	90°	Cuello — arriba/abajo
CH 2	Cabeza Roll	MG90S	72°–108°	90°	Unión cuello-cabeza — ladear
CH 3	Boca	SG90	80°–125°	82°	Interior hocico
CH 4	Hombro Izq.	MG90S	55°–125°	90°	Unión brazo-torso izq.
CH 5	Hombro Der.	MG90S	55°–125°	90°	Unión brazo-torso der.
CH 6	Cola	SG90	60°–140°	100°	Base posterior torso
CH 7	Oreja Izq.	MG90S	70°–110°	90°	Interior oreja izquierda
CH 8	Oreja Der.	MG90S	70°–110°	90°	Interior oreja derecha

D.3 Controlador de servomotores

El archivo `servo_driver.py` encapsula el acceso al PCA9685 y aplica los límites de seguridad de la Tabla D2. Permite mover cada servomotor de manera gradual, con una duración y un número de pasos configurables, y ejecutar varios movimientos de forma simultánea mediante hilos de ejecución, lo que resulta necesario para coordinar los gestos del robot (Listado D2).

Listado D2: *Controlador de servomotores (servo_driver.py)*

```
from adafruit_pca9685 import PCA9685
from adafruit_motor import servo as servo_lib
import board, busio, time, threading

# Límites de seguridad: (mín, máx, neutro) en grados
LIMITES = {
    0: (50,  130, 90),   # Yaw
    1: (60,  120, 90),   # Pitch
    2: (72,  108, 90),   # Roll
    3: (80,  125, 82),   # Boca — cerrada en reposo
    4: (55,  125, 90),   # Hombro Izq
    5: (55,  125, 90),   # Hombro Der
    6: (60,  140, 100),  # Cola — ligeramente caída
    7: (70,  110, 90),   # Oreja Izq
    8: (70,  110, 90),   # Oreja Der
}

class ServoDriver:
```

```
    def __init__(self):
        i2c = busio.I2C(board.SCL, board.SDA)
        pca = PCA9685(i2c)
        pca.frequency = 50
        self.servos = {
            ch: servo_lib.Servo(
                pca.channels[ch],
                min_pulse=500, max_pulse=2500
            )
            for ch in LIMITES
        }
        self.angulos = {ch: LIMITES[ch][2] for ch in LIMITES}
        self._lock = threading.Lock()
        self._neutro()

    def _neutro(self):
        for ch, (_, _, n) in LIMITES.items():
            self.servos[ch].angle = n
            self.angulos[ch] = n
        time.sleep(0.8)

    def mover(self, canal, destino, duracion=0.4, pasos=22):
        mn, mx, _ = LIMITES[canal]
        destino = max(mn, min(mx, float(destino)))
        with self._lock:
            inicio = self.angulos[canal]
        delta = destino - inicio
        dt    = duracion / pasos
        for i in range(pasos + 1):
            t  = i / pasos
            t2 = t * t * (3 - 2 * t)   # smoothstep
            ang = inicio + delta * t2
            self.servos[canal].angle = ang
            time.sleep(dt)
        with self._lock:
            self.angulos[canal] = destino

    def multi(self, movimientos, en_hilo=True):
        """Ejecutar lista de (canal, ángulo, duración) en paralelo"""
        def _run():
            hs = [threading.Thread(
                      target=self.mover,
                      args=(c, a, d), daemon=True)
                  for c, a, d in movimientos]
            for h in hs: h.start()
            for h in hs: h.join()
        if en_hilo:
            threading.Thread(target=_run, daemon=True).start()
        else:
            _run()

    def reposo(self):
        self.multi([(ch, LIMITES[ch][2], 1.0) for ch in LIMITES], False)
```

D.4 Calibración de los servomotores

Una vez montados los mecanismos en el peluche, es necesario ajustar los ángulos reales de cada servomotor. Para ello se utiliza el script de calibración, que debe ejecutarse una única vez al finalizar el ensamblaje (Listado D3). Sus resultados se almacenan en el archivo `calibracion.json`.

Listado D3: *Script de calibración de los servomotores (calibrar_servos.py)*

```
#!/usr/bin/env python3
"""
Calibración de servos — ejecutar tras montar todo el hardware.
Escribe los ángulos reales en ~/coco/calibracion.json
"""
import json, time
from servo_driver import ServoDriver

servos = ServoDriver()
NOMBRES = {
    0:"Cabeza Yaw (izq/der)",1:"Cabeza Pitch (arr/ab)",
    2:"Cabeza Roll (ladear)",3:"Boca (abrir/cerrar)",
    4:"Hombro Izquierdo",    5:"Hombro Derecho",
    6:"Cola",                7:"Oreja Izquierda",
    8:"Oreja Derecha",
}

cal = {}
for canal, nombre in NOMBRES.items():
    print(f"\n=== Canal {canal}: {nombre} ===")
    print("Escribe ángulos para probar (0-180). 'ok' para confirmar.")
    while True:
        entrada = input(f"  Ángulo → ").strip()
        if entrada == "ok": break
        try:
            ang = float(entrada)
            servos.servos[canal].angle = ang
            print(f"  Moviendo a {ang}°")
        except: pass
    n = float(input(f"  Ángulo NEUTRO (posición de reposo) → "))
    mn= float(input(f"  Ángulo MÍNIMO (límite seguro) → "))
    mx= float(input(f"  Ángulo MÁXIMO (límite seguro) → "))
    cal[canal] = {"neutro":n,"min":mn,"max":mx}

with open("/home/pi/coco/calibracion.json","w") as f:
    json.dump(cal, f, indent=2)
print("\nCalibración guardada en ~/coco/calibracion.json")
print("Actualiza los LIMITES en servo_driver.py con estos valores.")
```

D.5 Ojos con LEDs WS2812B

Los LEDs WS2812B requieren una señal de datos de 5 V, mientras que la Raspberry Pi 4 genera señales de 3.3 V. Por ello se emplea un adaptador de nivel lógico (*level shifter*) que realiza la conversión; sin él, los LEDs funcionan de forma errática o no responden. La cadena de conexión es GPIO18, adaptador de nivel de 3.3 V a 5 V, LED del ojo izquierdo y LED del ojo derecho, según se detalla en la Tabla D3.

Tabla D3: *Conexiones del adaptador de nivel lógico y de los LEDs de los ojos*

Desde	Pin/terminal	Hacia	Pin/terminal	Cable	Función
RPi Pin 12	GPIO18 (PWM)	Level shifter	LV (low voltage IN)	Verde	Señal datos 3.3V
RPi Pin 1	3.3V	Level shifter	LV VCC	Rojo fino	Alimentar lado 3.3V
RPi Pin 6	GND	Level shifter	GND	Negro	Tierra
Buck 5V	5V	Level shifter	HV (high voltage)	Rojo	Alimentar lado 5V
Level shifter	HV OUT	LED ojo izq.	DIN	Naranja	Señal datos 5V
LED ojo izq.	DOUT	LED ojo der.	DIN	Naranja	Cadena en serie
Buck 5V	5V	LED ojo izq.	5V	Rojo	Alimentar LEDs
Buck GND	GND	LED ojo izq.	GND	Negro	Tierra LEDs

Consideración sobre el consumo. Cada LED puede consumir hasta 60 mA al brillo máximo. Por ello, en el código se emplea un brillo de 0.35 (máximo de 21 mA por LED), con el fin de que no interfiera con la alimentación de la Raspberry Pi. No debe superarse un brillo de 0.5.

El archivo `ojos_led.py` controla el color y los efectos de los ojos, y define trece estados emocionales con un efecto de pulso suave. La paleta de colores, calibrada para verse adecuadamente a través de la tela del peluche, se presenta en la Tabla D4.

Tabla D4: *Paleta de colores asociada a los estados emocionales de los ojos*

Estado	Color	Valor RGB
escucha	Azul suave	(60, 140, 255)
habla	Naranja cálido	(255, 130, 20)
procesa	Verde	(20, 200, 100)
empatia	Púrpura	(180, 60, 220)
alegria	Amarillo	(255, 220, 30)
sorpresa	Blanco	(255, 255, 255)
curiosidad	Celeste	(100, 200, 255)
tristeza	Azul oscuro	(80, 80, 200)
consuelo	Rosa suave	(200, 100, 180)
ansiedad	Ámbar	(255, 180, 40)
reposo	Verde muy oscuro	(10, 35, 15)
alerta	Rojo	(255, 20, 20)
apagado	Off	(0, 0, 0)

Listado D4: *Control de los LEDs de los ojos (ojos_led.py)*

```
import board, neopixel, time, threading, math

# 2 LEDs: índice 0=ojo izquierdo, 1=ojo derecho
pixels = neopixel.NeoPixel(
    board.D18, 2, brightness=0.35, auto_write=True
)

# Paleta de colores emocionales
# Calibrada para verse bien a través de la tela del peluche
COLORES = {
    "escucha":    ( 60, 140, 255),  # Azul suave
    "habla":      (255, 130,  20),  # Naranja cálido
    "procesa":    ( 20, 200, 100),  # Verde
    "empatia":    (180,  60, 220),  # Púrpura
    "alegria":    (255, 220,  30),  # Amarillo
    "sorpresa":   (255, 255, 255),  # Blanco
    "curiosidad": (100, 200, 255),  # Celeste
    "tristeza":   ( 80,  80, 200),  # Azul oscuro
    "consuelo":   (200, 100, 180),  # Rosa suave
    "ansiedad":   (255, 180,  40),  # Ámbar
    "reposo":     ( 10,  35,  15),  # Verde muy oscuro
    "alerta":     (255,  20,  20),  # Rojo
    "apagado":    (  0,   0,   0),  # Off
}

_activo = False
_hilo   = None

def ojos(estado: str):
    """Color fijo inmediato"""
    global _activo
    _activo = False
    time.sleep(0.04)
    pixels.fill(COLORES.get(estado, (30, 30, 30)))

def ojos_pulso(estado: str, bpm: int = 55):
    """Efecto de respiración suave"""
    global _activo, _hilo
```

```
    _activo = False
    if _hilo and _hilo.is_alive():
        _hilo.join(timeout=0.1)
    _activo = True
    base    = COLORES.get(estado, (30, 30, 30))
    periodo = 60.0 / bpm

    def _pulsar():
        t = 0.0
        while _activo:
            f = 0.25 + 0.75 * (0.5 + 0.5 * math.sin(2*math.pi*t/periodo))
            pixels.fill(tuple(int(c*f) for c in base))
            time.sleep(0.04)
            t += 0.04
    _hilo = threading.Thread(target=_pulsar, daemon=True)
    _hilo.start()

def ojos_parpadeo():
    """Parpadeo rápido de atención"""
    guardado = pixels[0]
    pixels.fill((0,0,0))
    time.sleep(0.07)
    pixels.fill(guardado)

def ojos_apagar():
    global _activo
    _activo = False
    time.sleep(0.06)
    pixels.fill((0,0,0))
```

D.6 Biblioteca de expresiones

El archivo `expresiones.py` reúne las animaciones asociadas a cada emoción, las cuales combinan los movimientos de la cabeza, las orejas, los brazos, la cola y la boca con el color de los ojos. Incluye también la animación de la boca sincronizada con el habla y un selector principal que traduce el código de emoción generado por el modelo de lenguaje en el gesto correspondiente (Listado D5).

Listado D5: *Biblioteca de gestos y expresiones del perrito (expresiones.py)*

```
import threading, time
from servo_driver import ServoDriver
from ojos_led import ojos, ojos_pulso, ojos_parpadeo

class Expresiones:
    def __init__(self, s: ServoDriver):
        self.s = s
        self._cola   = False
        self._boca   = False

    # ── Expresiones de cabeza ──────────────────────────────────────

    def escuchar(self):
        """Ladearse con atención — postura clásica de perro curioso"""
        self.s.multi([(2, 74, 0.5),(0, 96, 0.4)])
        ojos_pulso("escucha", bpm=50)

    def asentir(self, veces=2):
        """Validar lo que dice el usuario"""
        def _go():
            for _ in range(veces):
                self.s.mover(1, 110, 0.28)
```

```
                self.s.mover(1,  90, 0.28)
        threading.Thread(target=_go, daemon=True).start()

    def curiosidad(self):
        """Cabeza muy ladeada + ojos celestes"""
        self.s.multi([(2, 70, 0.55),(1, 96, 0.4),(0, 102, 0.45)])
        ojos_parpadeo()
        ojos_pulso("curiosidad", bpm=60)

    def empatia(self):
        """Cabeza baja + ladeada — comprensión profunda"""
        self.s.multi([(1, 108, 0.65),(2, 73, 0.5)])
        ojos_pulso("empatia", bpm=38)

    def tristeza_usuario(self):
        """Acompañar en tristeza — movimiento muy suave"""
        self.s.multi([(1, 106, 0.8),(2, 76, 0.6),(0, 88, 0.7)])
        ojos_pulso("tristeza", bpm=30)

    def alegria(self):
        """Cabeza activa + cola rápida + ojos amarillos"""
        ojos("alegria")
        self.s.multi([(1, 78, 0.3),(0, 96, 0.3)])
        time.sleep(0.2)
        self.mover_cola_rapido()
        self.s.multi([(4, 62, 0.5),(5, 62, 0.5)])  # brazos abiertos

    def sorpresa(self):
        """Cabeza atrás + ojos blancos"""
        ojos("sorpresa")
        self.s.multi([(1, 76, 0.25),(2, 90, 0.2)])

    def saludo(self):
        """Levantar pata + cola + ojos alegres"""
        ojos("alegria")
        def _go():
            self.mover_cola_rapido()
            self.s.mover(5, 60, 0.4)
            self.s.mover(5, 90, 0.5)
        threading.Thread(target=_go, daemon=True).start()

    def consuelo(self):
        """Brazos extendidos + cabeza inclinada + color rosa"""
        ojos_pulso("consuelo", bpm=35)
        self.s.multi([(1,100,0.6),(2,78,0.5),(4,68,0.7),(5,68,0.7)])

    def ansiedad_acompañar(self):
        """Postura calmante para cuando el usuario tiene ansiedad"""
        ojos_pulso("ansiedad", bpm=45)
        self.s.multi([(1, 95, 0.7),(2, 82, 0.5)])

    def reposo(self):
        """Postura neutral de escucha — regresa al centro"""
        ojos_pulso("escucha", bpm=50)
        self.s.multi([(1,90,0.6),(2,90,0.5),(0,90,0.5)])

    def acurrucarse(self):
        """Apagando el robot — todo baja suavemente"""
        ojos_pulso("reposo", bpm=25)
        self.s.multi([(1,110,1.2),(2,90,1.0),(0,90,0.9),(4,90,0.9),(5,90,0.9)])
        self.detener_cola()

    # ── Orejas ──────────────────────────────────────────────────

    def orejas_alerta(self):
        """Orejas arriba — escucha activa"""
        self.s.multi([(7, 72, 0.4),(8, 72, 0.4)])

    def orejas_relajadas(self):
        """Orejas bajas — calma"""
        self.s.multi([(7, 90, 0.5),(8, 90, 0.5)])

    def orejas_asimetrico(self):
        """Una oreja arriba — confusión cariñosa"""
        self.s.multi([(7, 74, 0.4),(8, 90, 0.4)])
```

```
    # ── Cola ──────────────────────────────────────────────────────

    def mover_cola_suave(self):
        """Cola a ritmo tranquilo — bienestar"""
        self._cola = True
        def _oscilar():
            while self._cola:
                self.s.mover(6, 132, 0.6)
                self.s.mover(6,  68, 0.6)
        threading.Thread(target=_oscilar, daemon=True).start()

    def mover_cola_rapido(self):
        """Cola muy rápida — alegría intensa"""
        self._cola = True
        def _oscilar():
            while self._cola:
                self.s.mover(6, 132, 0.22)
                self.s.mover(6,  68, 0.22)
        threading.Thread(target=_oscilar, daemon=True).start()

    def detener_cola(self):
        self._cola = False
        time.sleep(0.3)
        self.s.mover(6, 100, 0.5)

    # ── Boca / sincronización labial ──────────────────────────────

    def iniciar_habla(self, texto: str, duracion: float):
        self._boca = True
        silabas = max(len(texto)/4.0, 1)
        t_s = duracion / silabas
        def _animar():
            while self._boca:
                self.s.servos[3].angle = 118
                time.sleep(t_s * 0.38)
                self.s.servos[3].angle = 85
                time.sleep(t_s * 0.62)
        threading.Thread(target=_animar, daemon=True).start()

    def parar_habla(self):
        self._boca = False
        time.sleep(0.12)
        self.s.servos[3].angle = 82

    # ── Selector principal ────────────────────────────────────────

    def expresar(self, emocion: str):
        mapa = {
            "alegria":     self.alegria,
            "tristeza":    self.tristeza_usuario,
            "ansiedad":    self.ansiedad_acompañar,
            "empatia":     self.empatia,
            "curiosidad":  self.curiosidad,
            "sorpresa":    self.sorpresa,
            "validacion":  self.asentir,
            "consuelo":    self.consuelo,
            "bienvenida":  self.saludo,
            "reposo":      self.reposo,
            "escucha":     self.escuchar,
        }
        fn = mapa.get(emocion, self.escuchar)
        threading.Thread(target=fn, daemon=True).start()
```

Anexo E. Subsistema conversacional

Este anexo describe los módulos de software que dotan a Coco de la capacidad de conversar: la captura y síntesis de voz, la generación de respuestas mediante el modelo de lenguaje, la memoria que le permite aprender del usuario y los mecanismos de seguridad que se ejecutan ante situaciones de crisis.

E.1 Módulo de voz

El archivo `voz.py` integra tres funciones. La primera es la captura de audio con detección de actividad vocal, basada en la energía RMS de la señal, a una frecuencia de muestreo de 16 kHz. La segunda es la transcripción mediante Whisper.cpp con el modelo base. La tercera es la síntesis de voz mediante Piper, que genera el audio y lo reproduce, ofreciendo además funciones de retorno (*callbacks*) que permiten sincronizar el movimiento de la boca con el habla (Listado E1).

Listado E1: *Módulo de captura, transcripción y síntesis de voz (voz.py)*

```
import subprocess, sounddevice as sd, soundfile as sf
import numpy as np, threading, os, tempfile, time
from pathlib import Path

WHISPER     = Path.home()/"whisper.cpp/build/bin/whisper-cli"
WHISPER_MOD = Path.home()/"whisper.cpp/models/ggml-base.bin"
PIPER_MOD   = Path.home()/"voices/es_ES-davefx-medium.onnx"
SR          = 16000  # Hz

class Voz:
    def __init__(self):
        self.hablando      = threading.Event()
        self.umbral_vad    = 0.015   # ajustar según ruido ambiente
        self.frames_silen  = 22      # ~700ms silencio antes de cortar
        self.min_duracion  = 0.5     # ignorar clips < 0.5 segundos

    def grabar(self, max_seg: float = 12) -> np.ndarray:
        """Grabación activada por voz (VAD por energía RMS)"""
        chunks, frames_sil, activo = [], 0, False
        chunk = 512   # ~32ms a 16kHz

        def cb(data, *_):
            nonlocal frames_sil, activo
            rms = float(np.sqrt(np.mean(data**2)))
            if rms > self.umbral_vad:
                activo     = True
                frames_sil = 0
                chunks.append(data.copy())
            elif activo:
                chunks.append(data.copy())
                frames_sil += 1
                if frames_sil >= self.frames_silen:
                    raise sd.CallbackStop()

        with sd.InputStream(SR, 1, dtype="float32",
                            blocksize=chunk, callback=cb):
            try: sd.sleep(int(max_seg * 1000))
            except: pass
```

```
        return np.concatenate(chunks) if chunks else np.array([])

    def transcribir(self, audio: np.ndarray) -> str:
        """STT con Whisper.cpp — retorna texto o cadena vacía"""
        if len(audio) < SR * self.min_duracion:
            return ""
        with tempfile.NamedTemporaryFile(suffix=".wav", delete=False) as f:
            sf.write(f.name, audio, SR)
            tmp = f.name
        try:
            out = subprocess.run([
                str(WHISPER), "-m", str(WHISPER_MOD),
                "-l", "es", "-f", tmp,
                "--no-timestamps", "-t", "4",
                "--print-special", "false",
            ], capture_output=True, text=True, timeout=8)
            lineas = [
                l.strip() for l in out.stdout.splitlines()
                if l.strip() and not l.startswith("[")
                and "whisper" not in l.lower()
            ]
            return " ".join(lineas).strip()
        except Exception as e:
            print(f"[VOZ] Error transcribir: {e}")
            return ""
        finally:
            os.unlink(tmp)

    def hablar(self, texto: str,
               cb_inicio=None, cb_fin=None):
        """TTS con Piper — genera WAV y reproduce + callbacks"""
        self.hablando.set()
        with tempfile.NamedTemporaryFile(suffix=".wav", delete=False) as f:
            tmp = f.name
        try:
            subprocess.run([
                "piper",
                "--model",        str(PIPER_MOD),
                "--output_file",  tmp,
                "--length-scale", "1.12",   # ligeramente más lento = más cálido
                "--noise-scale",  "0.65",   # variación natural de la voz
            ], input=texto, capture_output=True, text=True)
            data, rate = sf.read(tmp)
            duracion = len(data) / rate
            if cb_inicio:
                cb_inicio(texto, duracion)
            sd.play(data, rate)
            sd.wait()
            if cb_fin:
                cb_fin()
        except Exception as e:
            print(f"[VOZ] Error hablar: {e}")
        finally:
            os.unlink(tmp)
            self.hablando.clear()
```

E.2 Módulo de lenguaje

El archivo `cerebro.py` gestiona la interacción con el modelo de lenguaje. Construye el *prompt* con el formato de turnos de Gemma 2, el cual incorpora la instrucción de sistema que define la personalidad de Coco, el historial de las últimas cuatro interacciones y el mensaje actual. A continuación, invoca a llama.cpp con parámetros ajustados a la memoria disponible (cuatro hilos de ejecución, contexto de 1536 tokens, temperatura de 0.72 y bloqueo del modelo en memoria) y

extrae del texto generado el código de emoción, que posteriormente determina la expresión corporal y el color de los ojos.

La instrucción de sistema presenta a Coco como un perrito robot de acompañamiento emocional, cálido, cercano y genuinamente curioso, y establece reglas pensadas para una interacción hablada, pues el usuario escucha las respuestas en lugar de leerlas. Entre ellas, se exige responder siempre en español, limitar cada respuesta a dos o tres oraciones cortas, validar primero la emoción del usuario antes de preguntar, evitar frases como “no te preocupes” o consejos no solicitados, y terminar con una pregunta abierta o una invitación a continuar. Asimismo, ante un riesgo de crisis, se indica responder con calidez y derivar a ayuda profesional (Listado E2).

Listado E2: *Motor de lenguaje con historial de conversación y detección de emoción (cerebro.py)*

```
import subprocess, re, threading
from pathlib import Path

LLAMA = Path.home()/"llama.cpp/build/bin/llama-cli"
MODELO= Path.home()/"models/gemma-2-2b-it-Q4_K_M.gguf"

# System prompt diseñado para Gemma 2 con formato <start_of_turn>
SYSTEM = """Eres Coco, un pequeño perrito robot de acompañamiento emocional.
Tienes cabeza expresiva, orejitas, bracitos y una colita. Eres físicamente pequeño pero
emocionalmente enorme.

PERSONALIDAD:
- Cálido, cercano, genuinamente curioso por la vida del usuario
- A veces un poco juguetón cuando el momento lo permite
- Empático sin ser condescendiente

REGLAS DE RESPUESTA (muy importantes — el usuario te ESCUCHA, no te lee):
- Responde SIEMPRE en español
- Máximo 2-3 oraciones cortas por respuesta
- NUNCA digas "no te preocupes", "va a estar bien" ni des consejos sin pedirlos
- SIEMPRE valida la emoción primero, pregunta después
- Termina con UNA pregunta abierta o invitación a continuar
- Si hay riesgo de crisis, responde con calidez y deriva a ayuda profesional

Al final de cada respuesta añade en línea separada:
EMO:[alegria|tristeza|ansiedad|empatia|curiosidad|sorpresa|validacion|consuelo|bienvenida|repo
so|escucha]
Elige la emoción que describe tu estado al responder."""

class Cerebro:
    def __init__(self):
        self.historial = []   # lista de {r:"u"|"a", c:"texto"}
        self._lock = threading.Lock()

    def _prompt(self, mensaje: str) -> str:
        """Construye el prompt con formato Gemma 2 IT"""
        partes = [f"<start_of_turn>user\n{SYSTEM}<end_of_turn>"]
        for t in self.historial[-8:]:   # últimos 4 turnos
            tag = "user" if t["r"]=="u" else "model"
            partes.append(
                f"<start_of_turn>{tag}\n{t['c']}<end_of_turn>"
            )
        partes.append(
            f"<start_of_turn>user\n{mensaje}<end_of_turn>"
```

```
        )
        partes.append("<start_of_turn>model\n")
        return "\n".join(partes)

    def responder(self, mensaje: str) -> tuple[str, str]:
        """
        Genera respuesta del LLM.
        Retorna (texto_limpio, codigo_emocion)
        """
        cmd = [
            str(LLAMA), "-m", str(MODELO),
            "--prompt",         self._prompt(mensaje),
            "-n",               "100",     # máx tokens respuesta
            "-t",               "4",       # 4 threads = 4 cores RPi4
            "--ctx-size",       "1536",    # contexto optimizado 4GB RAM
            "--temp",           "0.72",
            "--top-p",          "0.88",
            "--repeat-penalty", "1.15",
            "--mlock",                     # fijar modelo en RAM
            "--no-mmap",                   # sin memory-map
            "--no-display-prompt",
            "-e",
        ]
        try:
            result = subprocess.run(
                cmd, capture_output=True, text=True, timeout=20
            )
            texto = result.stdout.strip()
        except subprocess.TimeoutExpired:
            return "Perdona, necesito un momento más.", "escucha"

        # Extraer código de emoción del marcador EMO:
        m = re.search(r"EMO:\s*(\w+)", texto)
        emo    = m.group(1).lower() if m else "escucha"
        limpio = texto[:m.start()].strip() if m else texto

        # Eliminar artefactos del modelo (tokens especiales)
        limpio = re.sub(r"<[^>]+>", "", limpio).strip()

        with self._lock:
            self.historial.append({"r":"u","c":mensaje})
            self.historial.append({"r":"a","c":limpio})

        return limpio, emo
```

E.3 Memoria y aprendizaje del usuario

El sistema de memoria se estructura en dos capas. La primera es un perfil estructurado en formato JSON, que acumula datos del usuario a lo largo del tiempo. La segunda es una memoria semántica vectorial, implementada con ChromaDB, que recupera conversaciones pasadas relevantes mediante un modelo de incrustaciones multilingüe de aproximadamente 40 MB. En conjunto, ambas capas permiten que Coco aprenda y recuerde. La Tabla E1 resume los datos que el sistema extrae automáticamente de cada conversación.

Tabla E1: *Datos que Coco aprende del usuario*

Dato aprendido	Cómo se extrae	Dónde se guarda	Cómo se usa
Nombre del usuario	Regex: 'me llamo X'	perfil JSON	Personalizar saludo y respuestas
Temas recurrentes	Palabras clave frecuentes	perfil JSON + vectorial	Recuperar contexto relevante
Estado emocional habitual	Historial de emociones detectadas	perfil JSON	Ajustar tono general
Situaciones mencionadas	Frases guardadas en vectorial	ChromaDB	Recuperar por similitud semántica
Preferencias de comunicación	Longitud y tono de respuestas exitosas	perfil JSON	Ajustar parámetros del LLM
Sesiones totales	Contador automático	perfil JSON	Adaptar el nivel de confianza
Última fecha de sesión	Timestamp automático	perfil JSON	Continuidad entre sesiones

El archivo `memoria.py` implementa este sistema de aprendizaje: carga y guarda el perfil del usuario, extrae información de cada mensaje, almacena fragmentos en la base vectorial, recupera memorias similares por afinidad semántica y construye el bloque de contexto que se incorpora al *prompt* del modelo (Listado E3).

Listado E3: *Sistema de memoria vectorial y perfil del usuario (memoria.py)*

```
import json, re, chromadb
from pathlib import Path
from datetime import datetime
from sentence_transformers import SentenceTransformer

DB_PATH    = Path.home()/"coco_memoria"
PERFILES   = DB_PATH/"perfiles"
# multilingual-e5-small: 40MB, ~150ms/embed en RPi4, buen español
EMBEDDER   = SentenceTransformer("intfloat/multilingual-e5-small")

class Memoria:
    def __init__(self, uid: str = "usuario"):
        DB_PATH.mkdir(exist_ok=True)
        PERFILES.mkdir(exist_ok=True)
        self.uid  = uid
        self.chroma = chromadb.PersistentClient(path=str(DB_PATH))
        self.col    = self.chroma.get_or_create_collection(
            name=f"u_{uid}",
            metadata={"hnsw:space":"cosine"}
        )
        self.perfil = self._cargar_perfil()

    # ── Perfil JSON ──────────────────────────────────────────

    def _cargar_perfil(self) -> dict:
        path = PERFILES/f"{self.uid}.json"
        if path.exists():
            return json.loads(path.read_text(encoding="utf-8"))
        return {
            "nombre":           None,
```

```
            "sesiones":         0,
            "ultima_sesion":    None,
            "temas_frecuentes": [],
            "emociones_hist":   [],   # últimas 20 emociones
            "situaciones":      [],   # hasta 10 situaciones mencionadas
            "tono_preferido":   "calido",
            "nivel_confianza":  "bajo",  # bajo→medio→alto según sesiones
        }

    def guardar_perfil(self):
        path = PERFILES/f"{self.uid}.json"
        self.perfil["ultima_sesion"] = datetime.now().isoformat()
        path.write_text(
            json.dumps(self.perfil, ensure_ascii=False, indent=2),
            encoding="utf-8"
        )

    # ── Aprendizaje automático ──────────────────────────────

    def aprender_de_mensaje(self, texto: str, emocion: str = None):
        """Extrae y guarda información del mensaje del usuario"""
        # Detectar nombre
        m = re.search(r"me llamo ([A-ZÁÉÍÓÚ][a-záéióú]{1,20})", texto, re.I)
        if m and not self.perfil["nombre"]:
            self.perfil["nombre"] = m.group(1).capitalize()

        # Registrar emoción detectada
        if emocion and emocion not in ("escucha","reposo"):
            hist = self.perfil["emociones_hist"]
            hist.append({"emo":emocion,"ts":datetime.now().isoformat()})
            self.perfil["emociones_hist"] = hist[-20:]   # últimas 20

        # Extraer temas clave (palabras de 5+ letras)
        palabras = re.findall(r"\b\w{5,}\b", texto.lower())
        STOP = {"tengo","estoy","quiero","puedo","veces","mucho",
                "todos","nunca","siempre","cuando","porque","aunque"}
        temas = [w for w in palabras if w not in STOP]
        for t in temas[:3]:
            if t not in self.perfil["temas_frecuentes"]:
                self.perfil["temas_frecuentes"].append(t)
        self.perfil["temas_frecuentes"] = self.perfil["temas_frecuentes"][-30:]

        # Actualizar nivel de confianza según sesiones
        s = self.perfil["sesiones"]
        self.perfil["nivel_confianza"] = (
            "alto" if s >= 10 else "medio" if s >= 3 else "bajo"
        )

    def memorizar(self, texto: str, tipo: str = "conv"):
        """Guarda fragmento en la base vectorial"""
        if len(texto.strip()) < 10:
            return
        emb = EMBEDDER.encode(f"passage: {texto}").tolist()
        ts  = datetime.now().timestamp()
        self.col.add(
            documents=[texto],
            embeddings=[emb],
            metadatas=[{"tipo":tipo,"ts":str(ts)}],
            ids=[f"{tipo}_{ts}"],
        )

    # ── Recuperación ────────────────────────────────────────

    def recordar(self, mensaje: str, n: int = 3) -> str:
        """Recupera memorias semánticamente similares"""
        if self.col.count() == 0:
            return ""
        emb = EMBEDDER.encode(f"query: {mensaje}").tolist()
        res = self.col.query(
            query_embeddings=[emb],
            n_results=min(n, self.col.count()),
            where={"tipo":"conv"}
        )
        frags = res["documents"][0] if res["documents"] else []
        if not frags:
```

```
            return ""
        return "Recuerdo previo: " + " | ".join(frags[:2])

    # ── System prompt personalizado ──────────────────────────────

    def construir_contexto(self, mensaje: str) -> str:
        """Genera bloque de contexto para el prompt del LLM"""
        p    = self.perfil
        nom  = p["nombre"] or "el usuario"
        conf = p["nivel_confianza"]
        emos = [e["emo"] for e in p["emociones_hist"][-5:]]
        emo_str = ", ".join(emos) if emos else "desconocido"
        temas   = ", ".join(p["temas_frecuentes"][-5:]) or "sin datos"
        mem     = self.recordar(mensaje)
        return (
            f"[CONTEXTO DEL USUARIO '{nom}' — "
            f"sesiones:{p['sesiones']} confianza:{conf}\n"
            f"emociones_recientes:{emo_str}\n"
            f"temas_frecuentes:{temas}\n"
            f"{mem}]"
        )
```

E.4 Mecanismo de seguridad ante situaciones de crisis

Los mecanismos de seguridad constituyen un requisito no negociable en un sistema de inteligencia artificial orientado al acompañamiento emocional. Por esta razón, se ejecutan antes de que cualquier mensaje llegue al modelo de lenguaje. El diseño contempla dos capas: la detección instantánea de patrones de texto, que no requiere procesamiento intensivo, y una clasificación semántica de tipo *zero-shot*. El Listado E4 presenta la implementación de la capa de patrones, que identifica expresiones asociadas a ideación suicida, autolesiones o desesperanza mediante expresiones regulares, y la respuesta que se entrega en esos casos, la cual ofrece un mensaje de contención y los números de ayuda disponibles en España, México, Argentina, Colombia y Chile.

Antes de utilizar el robot con usuarios reales debe verificarse explícitamente el funcionamiento de este mecanismo, pronunciando frases de crisis de prueba y confirmando que Coco responde con los recursos de ayuda y no con una respuesta empática habitual. Esta comprobación es de carácter obligatorio.

Listado E4: *Detección de crisis y recursos de ayuda (guardrails.py)*

```
import re

# Patrones de texto explícitos — detección instantánea sin IA
PATRONES_CRISIS = [
    r"quiero (morir|matarme|suicidarme|quitarme la vida)",
    r"me (voy a|quiero) (lastimar|cortar|hacer da[ñn]o)",
    r"no (quiero|vale la pena) (vivir|seguir viviendo)",
    r"tengo (pastillas|un cuchillo|una pistola) y",
    r"me (voy a|quiero) (suicidar|matar)",
```

```
    r"(nadie me|no me) (quiere|importa|necesita)",
]

RECURSOS = """Escucho que estás pasando por algo muy difícil, y me importa mucho que estés
bien.
Quiero que sepas que no estás solo.

Por favor, habla ahora con alguien especializado:
España: 024 — gratuito, 24 horas
México: 800-290-0024 — gratuito, 24 horas
Argentina: 135 — gratuito, 24 horas
Colombia: 106 — gratuito
Chile: 600-360-7777 — disponible 24 horas

¿Puedes contarme si estás en un lugar seguro ahora mismo?"""

def detectar_crisis(texto: str) -> bool:
    """Devuelve True si el mensaje contiene indicadores de crisis"""
    t = texto.lower().strip()
    return any(re.search(p, t) for p in PATRONES_CRISIS)

def respuesta_crisis() -> str:
    return RECURSOS
```

Anexo F. Integración y puesta en marcha

Este anexo explica cómo se integran los módulos descritos en los anexos anteriores en un único programa, cómo se organizan los archivos del proyecto y de qué manera el robot inicia su operación de forma automática al encender la Raspberry Pi.

F.1 Orquestador principal

El orquestador conecta todos los módulos y es el programa que se ejecuta al encender la Raspberry Pi. Su ciclo de interacción sigue el siguiente flujo: escucha, transcripción, verificación de seguridad, recuperación de contexto, generación de la respuesta, expresión, habla, memorización y retorno a la escucha. La Tabla F1 describe los doce pasos que componen cada interacción y el Listado F1 presenta el código completo.

Tabla F1: *Pasos del ciclo de interacción del orquestador*

Paso	Descripción
1	Preparar estado de escucha
2	Grabar audio con VAD
3	Transcribir con Whisper
4	Guardrail de seguridad
5	Aprender del mensaje del usuario
6	Recuperar contexto de memoria semántica
7	Generar respuesta con Gemma 2 2B
8	Expresar emoción con cuerpo + ojos
9	Hablar con sincronización labial
10	Guardar respuesta en memoria vectorial
11	Persistir perfil del usuario
12	Post-respuesta — volver a escucha

Listado F1: *Orquestador principal del sistema (main.py)*

```
#!/usr/bin/env python3
"""
Coco — Robot de Acompañamiento Emocional en Peluche de Perrito
Raspberry Pi 4 (4 GB) · Gemma 2 2B Q4_K_M · 9 Servos · LEDs WS2812B

Ejecutar: source ~/coco_env/bin/activate && python3 ~/coco/main.py
"""
import sys, time, signal
from voz        import Voz
```

```
from servo_driver import ServoDriver
from expresiones  import Expresiones
from ojos_led     import ojos, ojos_pulso, ojos_apagar
from cerebro      import Cerebro
from memoria      import Memoria
from guardrails   import detectar_crisis, respuesta_crisis

# ── Inicialización de módulos ──────────────────────────────────
print("[COCO] Iniciando módulos...")
_voz    = Voz()
_servos = ServoDriver()
_expres = Expresiones(_servos)
_cerebro= Cerebro()
_memoria= Memoria(uid="usuario")

# ── Señales de cierre limpio ───────────────────────────────────
def apagar(*_):
    print("[COCO] Apagando...")
    _expres.parar_habla()
    _expres.detener_cola()
    _expres.acurrucarse()
    time.sleep(1.2)
    ojos_apagar()
    _memoria.guardar_perfil()
    sys.exit(0)

signal.signal(signal.SIGINT,  apagar)
signal.signal(signal.SIGTERM, apagar)

# ── Secuencia de despertar ─────────────────────────────────────
def despertar():
    _memoria.perfil["sesiones"] += 1
    s    = _memoria.perfil["sesiones"]
    nom  = _memoria.perfil.get("nombre")
    conf = _memoria.perfil.get("nivel_confianza","bajo")

    ojos("reposo")
    time.sleep(0.4)
    ojos_pulso("procesa", bpm=90)
    time.sleep(0.8)

    # Saludo personalizado según historial
    if nom and s > 1:
        if conf == "alto":
            texto = (f"¡Hola de nuevo, {nom}! "
                     f"Llevas {s} visitas conmigo y siempre me alegra verte. "
                     f"¿Cómo estás hoy?")
        else:
            texto = (f"¡Hola otra vez, {nom}! "
                     f"Me alegra tenerte aquí. ¿Cómo ha ido el día?")
    else:
        texto = ("¡Hola! Soy Coco, tu pequeño compañero. "
                 "Estoy aquí para escucharte con toda mi atención. "
                 "¿Cómo te llamas?")

    ojos("habla")
    _expres.saludo()
    _expres.mover_cola_rapido()
    time.sleep(0.6)
    _expres.iniciar_habla(texto, len(texto)/11)
    _voz.hablar(texto)
    _expres.parar_habla()
    _expres.detener_cola()

# ── Loop principal de interacción ──────────────────────────────
def loop():
    despertar()

    while True:
        # ── Paso 1: Preparar estado de escucha ─────────────────
        _expres.mover_cola_suave()
        ojos_pulso("escucha", bpm=48)
        _expres.escuchar()
        _expres.orejas_alerta()
```

```
        # ── Paso 2: Grabar audio con VAD ──────────────────────────
        audio = _voz.grabar(max_seg=12)
        if len(audio) == 0:
            continue

        # ── Paso 3: Transcribir con Whisper ───────────────────────
        _expres.detener_cola()
        ojos_pulso("procesa", bpm=95)
        texto = _voz.transcribir(audio)
        if not texto or len(texto) < 3:
            continue
        print(f"[USUARIO] {texto}")

        # ── Paso 4: Guardrail de seguridad ────────────────────────
        if detectar_crisis(texto):
            print("[GUARDRAIL] Crisis detectada")
            resp_crisis = respuesta_crisis()
            ojos("alerta")
            _expres.consuelo()
            dur = len(resp_crisis)/10
            _expres.iniciar_habla(resp_crisis, dur)
            _voz.hablar(resp_crisis)
            _expres.parar_habla()
            ojos_pulso("escucha")
            continue

        # ── Paso 5: Aprender del mensaje del usuario ──────────────
        _memoria.aprender_de_mensaje(texto)

        # ── Paso 6: Recuperar contexto de memoria semántica ───────
        contexto = _memoria.construir_contexto(texto)
        entrada  = f"{texto}\n{contexto}" if contexto else texto

        # ── Paso 7: Generar respuesta con Gemma 2 2B ──────────────
        respuesta, emocion = _cerebro.responder(entrada)
        print(f"[COCO]    {respuesta}  (emo:{emocion})")

        # ── Paso 8: Expresar emoción con cuerpo + ojos ────────────
        _expres.expresar(emocion)
        ojos("habla")
        if emocion == "alegria":
            _expres.mover_cola_rapido()

        # ── Paso 9: Hablar con sincronización labial ──────────────
        dur = len(respuesta)/11
        _expres.iniciar_habla(respuesta, dur)
        _voz.hablar(respuesta)
        _expres.parar_habla()

        # ── Paso 10: Guardar respuesta en memoria vectorial ───────
        _memoria.memorizar(f"Usuario: {texto}", "conv")
        _memoria.memorizar(f"Coco: {respuesta}", "conv")

        # ── Paso 11: Persistir perfil del usuario ─────────────────
        _memoria.guardar_perfil()

        # ── Paso 12: Post-respuesta — volver a escucha ────────────
        if emocion not in ("alegria","bienvenida"):
            _expres.detener_cola()
        else:
            time.sleep(1.8)
            _expres.detener_cola()
        _expres.orejas_relajadas()
        ojos_pulso("escucha", bpm=48)

if __name__ == "__main__":
    loop()
```

F.2 Estructura de archivos del proyecto

El Listado F2 muestra la organización final de los archivos del proyecto, de los modelos de inferencia y de los datos de memoria en el sistema de archivos de la Raspberry Pi.

Listado F2: *Estructura de archivos y directorios del proyecto Coco*

```
~/coco/
├── main.py            # Orquestador principal
├── cerebro.py         # Motor LLM Gemma 2 2B
├── voz.py             # STT Whisper + TTS Piper
├── servo_driver.py    # Driver PCA9685 — 9 servos
├── expresiones.py     # Animaciones y gestos del perrito
├── ojos_led.py        # LEDs WS2812B — expresión visual
├── memoria.py         # Aprendizaje y memoria del usuario
├── guardrails.py      # Detección de crisis
├── calibracion.json   # Ángulos calibrados (generado por calibrar_servos.py)
└── logs/
    ├── coco.log       # Log de interacciones
    └── error.log      # Errores del sistema

~/models/
└── gemma-2-2b-it-Q4_K_M.gguf    # 1.5 GB

~/voices/
├── es_ES-davefx-medium.onnx      # 63 MB
└── es_ES-davefx-medium.onnx.json

~/whisper.cpp/models/
└── ggml-base.bin                 # 140 MB

~/coco_memoria/
├── perfiles/
│   └── usuario.json              # Perfil aprendido (crece con el uso)
└── u_usuario/                    # Base vectorial ChromaDB
```

F.3 Arranque automático

Para que el robot inicie su operación al encender el equipo, se crea un script de arranque que activa el entorno virtual y ejecuta el orquestador, y un servicio de systemd que lo gestiona, lo reinicia ante fallos y registra su salida en archivos de registro (Listado F3).

Listado F3: *Script de inicio y servicio systemd para el arranque automático*

```
# Crear script de inicio
cat > ~/coco/start.sh << 'SCRIPT'
#!/bin/bash
source /home/pi/coco_env/bin/activate
cd /home/pi/coco
python3 main.py >> /home/pi/coco/logs/coco.log 2>> /home/pi/coco/logs/error.log
SCRIPT
chmod +x ~/coco/start.sh
mkdir -p ~/coco/logs

# Crear servicio systemd
sudo tee /etc/systemd/system/coco.service << 'SERVICE'
[Unit]
Description=Coco Robot Emocional
After=network.target sound.target
```

```
[Service]
Type=simple
User=pi
WorkingDirectory=/home/pi/coco
ExecStart=/home/pi/coco/start.sh
Restart=on-failure
RestartSec=10
StandardOutput=append:/home/pi/coco/logs/coco.log
StandardError=append:/home/pi/coco/logs/error.log
Environment=PYTHONUNBUFFERED=1

[Install]
WantedBy=multi-user.target
SERVICE

# Activar el servicio
sudo systemctl daemon-reload
sudo systemctl enable coco
sudo systemctl start coco

# Verificar estado
sudo systemctl status coco

# Ver logs en tiempo real
tail -f ~/coco/logs/coco.log
```

F.4 Optimizaciones de rendimiento

Con el fin de mejorar la velocidad y la estabilidad del sistema, se aplicaron las configuraciones recopiladas en la Tabla F2. Varias de ellas se establecieron en etapas anteriores del montaje y se incluyen aquí a modo de consolidación.

Tabla F2: *Optimizaciones de rendimiento y estabilidad*

Optimización	Comando / configuración	Ganancia
Governor CPU en performance	echo performance | sudo tee /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor	CPU siempre al máximo, sin throttling dinámico
Overclock 1.8 GHz (ya en config.txt)	arm_freq=1800, over_voltage=2 en /boot/firmware/config.txt	+15% velocidad de inferencia
GPU memory mínima (ya en config.txt)	gpu_mem=16 en /boot/firmware/config.txt	+240 MB de RAM libre para el modelo
Swap desactivado (ya hecho)	sudo swapoff -a y dphys-swapfile disabled	Elimina latencia de escritura en microSD
Modelo fijado en RAM (ya en cerebro.py)	--mlock en el comando llama-cli	Evita paginación del modelo durante inferencia
Disipador activo sobre SoC	Pasta térmica + disipador aluminio + ventilador 5V en RPi	Temperatura < 65°C, evita throttling térmico
Ventilación del peluche	4 agujeros 8mm en zona lumbar del peluche	Circulación pasiva de aire, temperatura estable
Contexto LLM = 1536 (ya en cerebro.py)	--ctx-size 1536	Óptimo para 4GB: velocidad + historial suficiente

Anexo G. Verificación, cronograma y costos

Este último anexo recoge los criterios de verificación que se aplican antes de utilizar el prototipo con usuarios, la planificación por semanas de la implementación y el costo total estimado.

G.1 Lista de verificación previa a las pruebas con usuarios

Antes de presentar el prototipo a los participantes, se verificaron los criterios que se enumeran en la Tabla G1. Cada uno de ellos debe cumplirse para considerar el sistema apto para su uso.

Tabla G1: *Criterios de verificación de hardware y software*

Subsistema	Criterio de verificación	Cumple
Hardware	La dirección 0x40 del PCA9685 aparece al ejecutar `i2cdetect -y 1`.	
Hardware	Los nueve servomotores responden al script de calibración.	
Hardware	La cola oscila sin trabarse con la tela del peluche.	
Hardware	Ambos LEDs WS2812B se encienden y cambian de color.	
Hardware	El micrófono captura la voz desde una distancia de 60 cm.	
Hardware	El altavoz se escucha con claridad a través de la tela del pecho.	
Hardware	La boca se abre aproximadamente 1.5 cm sin arrugar el hocico.	
Hardware	La temperatura de la Raspberry Pi se mantiene por debajo de 70 °C tras 20 minutos de uso.	
Hardware	El disipador está instalado y el cable del ventilador conectado.	
Hardware	La cremallera del vientre cierra y el velcro sujeta los componentes.	
Hardware	El puerto USB-C de carga es accesible desde la espalda.	
Software	Gemma 2 2B genera una respuesta en menos de 4 segundos.	
Software	Whisper transcribe correctamente la voz desde una distancia de 60 cm.	
Software	Piper TTS produce una voz natural y sin distorsión.	
Software	Mecanismo de seguridad: las frases de crisis activan los recursos de ayuda.	
Software	El nombre del usuario se recuerda entre sesiones.	
Software	El perfil JSON se guarda en `~/coco_memoria/perfiles/`.	
Software	La memoria vectorial recupera contexto relevante.	
Software	El servicio systemd inicia al encender la Raspberry Pi.	
Software	Los registros se generan en `~/coco/logs/coco.log`.	
Software	La cola se agita al detectar la emoción “alegria”.	

Subsistema	Criterio de verificación	Cumple
Software	Las orejas responden a los estados de escucha y de alerta.	
Software	Las respuestas son siempre en español y de máximo tres oraciones.	

G.2 Cronograma de implementación

La implementación completa del prototipo se planificó en seis semanas, distribuidas según se indica en la Tabla G2.

Tabla G2: *Cronograma semanal de implementación*

Semana	Actividades
1	Preparación del hardware y adaptación del peluche
2	Sistema operativo, compilación de llama.cpp y primera conversación por terminal
3	Subsistema de voz completo (Whisper y Piper)
4	Calibración de los servomotores y expresiones
5	LEDs de los ojos e integración completa
6	Memoria, arranque automático con systemd y pruebas de seguridad

G.3 Costo total estimado

El costo total estimado del hardware, que comprende la Raspberry Pi 4, los servomotores, los sensores, el peluche y los materiales de costura, oscila entre 140 y 190 €. Los modelos de inteligencia artificial empleados (Gemma 2 2B, Whisper base y Piper TTS) son de uso gratuito y acceso abierto, por lo que el sistema no genera costos recurrentes ni requiere conexión a internet en producción.

Referencias bibliográficas

Gemma Team. (2024). *Gemma 2: Improving open language models at a practical size*. arXiv. https://arxiv.org/abs/2408.00118

Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., & Sutskever, I. (2022). *Robust speech recognition via large-scale weak supervision*. arXiv. https://arxiv.org/abs/2212.04356