Implementación de un Asistente de Inteligencia Artificial por Radio en Linux

Guía técnica de interfaz física, procesamiento de voz (STT–LLM–TTS) y automatización de PTT para radioaficionados

Volney Torres Almendro – LU1PVT

Resumen. Este documento describe cómo implementar un nodo o repetidor controlado desde una máquina Linux capaz de recibir transmisiones de voz en VHF/UHF FM, procesarlas mediante un modelo de lenguaje y responder automáticamente al aire respetando el protocolo de radioaficionado. La arquitectura toma como referencia la demostración realizada por el operador EA3JIE en 145,550 MHz FM con un transceptor Yaesu FTX-1 y el asistente de voz "Paola" con PTT automático y *roger beep* (video de referencia). Se detallan el aislamiento galvánico de la interfaz de hardware, las alternativas de software local e híbrido, un prototipo funcional en Python, los controles de seguridad de transmisión y el camino de integración sobre SVXLink.

1 Interfaz de hardware (PC Linux $\leftrightarrow$ Transceptor)

Para enlazar la placa de sonido de una computadora Linux con un transceptor de VHF/UHF sin introducir bucles de masa ni retornar radiofrecuencia hacia la computadora, es indispensable aislar galvánicamente las líneas de audio (RX y TX) y el circuito de disparo del PTT.

En un diseño discreto clásico (como la interfaz Digiface de EA3HYN), la etapa física se organiza en cuatro bloques funcionales resumidos en la Tabla 1.

Cuadro 1: Bloques funcionales de la interfaz aislada entre Linux y el transceptor.

Bloque	**Lado PC Linux**	**Lado Transceptor**	**Componentes clave y función**
Audio RX	Entrada de línea / Mic (`Line In`)	Salida de audio de accesorios	Transformador T1 (1:1, 600 Ω), ferritas anti-RF y potenciómetro multivuelta RV1 (4,7 kΩ) para nivelar sin saturar el ADC.
Audio TX	Salida de línea (`Line Out`)	Entrada de micrófono / datos	Transformador T2 (1:1, 600 Ω), ferritas anti-RF y potenciómetro RV2 (4,7 kΩ) para ajustar la desviación de FM sin sobremodulación.
Control de PTT	Adaptador USB–Serie (pines `RTS` o `DTR`)	Pin de PTT a masa	Resistencia limitadora R1 (560 Ω) excitando un optoacoplador (PC1) que conmuta el PTT a masa de forma aislada.
Detección de portadora (COS/COR)	Adaptador USB–Serie (pines `CTS` o `DCD`)	Salida lógica de *squelch* (COS/SQ)	Segundo optoacoplador que informa a Linux la apertura y el cierre exacto del *squelch*, evitando falsos disparos por QRM o ruido estático.

Mejora crítica mediante COS/COR: Depender únicamente de detección de actividad de voz por software (VAD) suele provocar grabaciones erróneas ante interferencias o ruido de banda. Incorporar la señal física de COS/SQ del transceptor hacia los pines `CTS` o `DCD` del puerto serie permite saber con precisión cuándo comienza y finaliza cada cambio.

Alternativas modernas: En equipos que dispongan de control CAT por USB o mediante interfaces comerciales compactas (como Digirig), es posible accionar el PTT por comandos serie utilizando `hamlib` (`rigctl`). Otra opción ampliamente adoptada en nodos AllStarLink y SVXLink consiste en utilizar el pin GPIO de un chip de audio USB `CM108`.

2 Arquitectura del pipeline de procesamiento en Linux

El flujo de datos para cada intercambio de radio opera como una cascada secuencial de seis pasos:

Audio RX (ALSA/PipeWire) → Trigger (COS / VAD) → STT → LLM → TTS → PTT + Audio TX + Roger Beep

Según el hardware disponible (especialmente memoria RAM y disponibilidad de GPU), el sistema puede desplegarse de manera totalmente autónoma sin conexión a Internet (ideal para redes de emergencia) o en modo híbrido para equipos livianos o placas como Raspberry Pi (Tabla 2).

Cuadro 2: Comparativa de componentes de software para despliegue local frente a arquitectura híbrida.

Etapa	**Opción 100% Offline (Local)**	**Opción Híbrida / Hardware liviano**	**Consideraciones técnicas**
Detección (Trigger)	Señal COS por `CTS`/`DCD`, o `webrtcvad` / `silero-vad`	Señal COS por `CTS`/`DCD` o `silero-vad`	En VHF FM, la señal COS por hardware elimina prácticamente la totalidad de los falsos disparos.
STT (Voz a texto)	`faster-whisper` o `whisper.cpp` (modelos `base` o `small` INT8)	`faster-whisper` (`tiny`/`base` en CPU) o API remota	`faster-whisper` utiliza CTranslate2 y corre entre 2 y 4 veces más rápido en CPU que la implementación original.
LLM (Cerebro)	Ollama (`qwen2.5:3b`, `llama3.2:3b` o `qwen3:4b-instruct` Q4)	API externa (Gemini / OpenAI) u Ollama 1,5B–3B	Definir `OLLAMA_KEEP_ALIVE=-1` para mantener el modelo en RAM. Evitar modelos de razonamiento largo (*thinking*) por su latencia.
TTS (Texto a voz)	Piper TTS (ONNX, voces `es_ES`, `es_AR`, `es_MX`)	`edge-tts` (requiere red; prosodia muy natural)	Piper sintetiza en CPU en fracciones de segundo sin Internet; `edge-tts` ofrece entonación más natural cuando hay conectividad.

Control de latencia y protocolo operativo: En comunicaciones por radio, las respuestas extensas monopolizan la frecuencia y aumentan el tiempo de espera del operador. El *system prompt* del modelo de lenguaje debe instruirlo explícitamente para responder en un

máximo de dos o tres oraciones breves, sin formato Markdown ni símbolos especiales, redactando cifras de manera pronunciable y finalizando siempre cada transmisión con la palabra “cambio”.

3 Implementación paso a paso en Linux

3.1 Dependencias del sistema y entorno Python

En distribuciones basadas en Debian o Ubuntu (equivalente con `dnf` en derivados de RHEL/Rocky Linux), se instalan las herramientas de audio, los permisos de acceso a los dispositivos serie y las bibliotecas de Python:

```
# System audio and serial tools
sudo apt update
sudo apt install -y python3-pip python3-venv alsa-utils sox ffmpeg portaudio19-dev

# Add user to dialout (serial PTT) and audio groups
sudo usermod -aG dialout,audio $USER

# Create virtual environment and install pipeline dependencies
python3 -m venv ~/radio-ai-venv
source ~/radio-ai-venv/bin/activate
pip install faster-whisper pyserial requests sounddevice soundfile numpy
```

Verificación inicial: Tras reiniciar la sesión para aplicar los permisos de grupo, ejecutar `arecord -l` y `ls /dev/ttyUSB*` para comprobar que el sistema operativo reconoce la interfaz de audio USB y el adaptador serie del PTT.

3.2 Configuración de síntesis local (Piper) y motor LLM (Ollama)

Se descarga el binario de Piper junto con una voz en español (`.onnx` y su archivo `.onnx.json`) y se mantiene Ollama ejecutándose en local con el modelo residente en memoria:

```
# Keep the model pinned in RAM to eliminate cold-start latency
export OLLAMA_KEEP_ALIVE=-1
ollama pull qwen2.5:3b

# Local speech synthesis test
echo "Estacion copiada, cambio." \
  | piper --model /opt/piper/es_ES-davefx-medium.onnx --output_file test.wav \
  && aplay test.wav
```

Confirmar que el audio de prueba salga por la tarjeta de sonido dedicada al transceptor y no por los altavoces integrados del equipo.

3.3 Demonio de control en Python

El siguiente prototipo implementa el ciclo completo de escucha y respuesta: espera la activación del *squelch* en el pin `CTS`, graba el audio entrante, realiza la transcripción local con `faster-whisper`, filtra por palabra de activación (*wake word*), consulta a Ollama en `localhost`, sintetiza la respuesta con Piper y controla el PTT mediante la línea `RTS` dentro de un bloque `try...finally` con límite de tiempo por software.

```
import subprocess
import time
import wave
import numpy as np
import requests
import serial
import sounddevice as sd
from faster_whisper import WhisperModel

# Hardware and audio configuration
SERIAL_PORT = "/dev/ttyUSB0"
AUDIO_DEVICE = None        # Set ALSA device index or name from sd.query_devices()
SAMPLE_RATE = 16000
MAX_RX_SECONDS = 25
MAX_TX_SECONDS = 45        # Software Time-Out Timer (TOT)
OLLAMA_URL = "http://localhost:11434/api/chat"
OLLAMA_MODEL = "qwen2.5:3b"
PIPER_BIN = "/usr/local/bin/piper"
PIPER_MODEL = "/opt/piper/es_ES-davefx-medium.onnx"

SYSTEM_PROMPT = (
    "Eres un asistente de radioaficionado operando en VHF FM. "
    "Responde en espanol claro, maximo 2 oraciones breves, sin formato markdown "
    "ni simbolos especiales. Termina siempre tu respuesta con la palabra cambio."
)

# Load STT model once into memory (int8 quantization for CPU efficiency)
stt_model = WhisperModel("base", device="cpu", compute_type="int8")
ser = serial.Serial(SERIAL_PORT, 9600, timeout=0.1)
ser.rts = False  # Ensure PTT is released at startup


def record_while_cos_active(filename="rx.wav"):
    """Wait for COS on serial CTS pin and record RX audio until squelch closes."""
    while not ser.cts:
        time.sleep(0.05)

    frames = []
    start_t = time.time()

    def callback(indata, frame_count, time_info, status):
        frames.append(indata.copy())

    with sd.InputStream(
        samplerate=SAMPLE_RATE,
        channels=1,
        dtype="int16",
        device=AUDIO_DEVICE,
        callback=callback,
    ):
        while ser.cts and (time.time() - start_t) < MAX_RX_SECONDS:
            time.sleep(0.05)

    if not frames:
        return None

    audio_data = np.concatenate(frames, axis=0)
    with wave.open(filename, "wb") as wf:
        wf.setnchannels(1)
        wf.setsampwidth(2)
        wf.setframerate(SAMPLE_RATE)
        wf.writeframes(audio_data.tobytes())
    return filename
```

```
def transcribe(filename):
    """Run local speech-to-text with Whisper."""
    segments, _ = stt_model.transcribe(filename, language="es", beam_size=1)
    return " ".join(seg.text.strip() for seg in segments).strip()


def ask_ollama(user_text):
    """Query local Ollama instance with radio-friendly constraints."""
    payload = {
        "model": OLLAMA_MODEL,
        "stream": False,
        "messages": [
            {"role": "system", "content": SYSTEM_PROMPT},
            {"role": "user", "content": user_text},
        ],
    }
    resp = requests.post(OLLAMA_URL, json=payload, timeout=30)
    resp.raise_for_status()
    return resp.json()["message"]["content"].strip()


def transmit_tts(text):
    """Synthesize speech with Piper, assert PTT via RTS, and play with TOT guard."""
    wav_out = "tx.wav"
    subprocess.run(
        [PIPER_BIN, "--model", PIPER_MODEL, "--output_file", wav_out],
        input=text.encode("utf-8"),
        check=True,
    )

    try:
        # Key transmitter and allow relay/PLL settling time
        ser.rts = True
        time.sleep(0.35)
        subprocess.run(["aplay", wav_out], timeout=MAX_TX_SECONDS, check=False)
        # Optional roger beep via sox: play -n synth 0.1 sine 1000 vol 0.3
        time.sleep(0.2)
    finally:
        # Always release PTT even on timeout or exception
        ser.rts = False


def main():
    print("Listening on VHF channel...")
    while True:
        rx_file = record_while_cos_active()
        if not rx_file:
            continue

        transcript = transcribe(rx_file)
        print(f"RX: {transcript}")

        # Wake-word filter so the assistant only replies when explicitly called
        if "paola" not in transcript.lower():
            continue

        reply = ask_ollama(transcript)
        print(f"TX: {reply}")
        transmit_tts(reply)


if __name__ == "__main__":
```

```
    try:
        main()
    finally:
        ser.rts = False
        ser.close()
```

3.4 Pruebas de laboratorio sin emisión al aire

Antes de conectar el transceptor a una antena exterior, realizar las pruebas iniciales utilizando un LED con resistencia limitadora en la salida del optoacoplador o conectando el equipo a una carga fantasma de 50 Ω. Ajustar la ganancia en `alsamixer` y los potenciómetros RV1/RV2 mientras se monitorea la transmisión desde un transceptor portátil cercano, verificando que la modulación resulte clara, con desviación adecuada y libre de distorsión o recortes.

4 Seguridad operativa, protección de RF y normativa

Advertencia de protección de etapa final (TOT) y cumplimiento reglamentario

Un bloqueo del proceso en Python o un error del adaptador USB–Serie puede mantener activado el PTT de forma indefinida, dañando los transistores finales del transceptor por exceso de ciclo de trabajo y bloqueando la frecuencia. Es obligatorio habilitar el **Time-Out Timer (TOT) por hardware en el propio transceptor** (entre 60 y 120 segundos como máximo) y considerar un circuito temporizador o *watchdog* físico adicional. Asimismo, toda estación automática o desatendida en bandas de radioaficionados debe ajustarse a la reglamentación vigente, incluyendo la identificación periódica con señal distintiva y la supervisión de las emisiones.

5 Integración avanzada en un repetidor con SVXLink

Para escalar desde un nodo simplex hacia un repetidor completo con cola de *squelch*, tonos subaudibles CTCSS e identificación periódica en telegrafía o voz, la vía recomendada en Linux es emplear **SVXLink** como controlador lógico principal del repetidor. Sobre esa base, el asistente de inteligencia artificial puede integrarse de dos maneras:

- **Módulo dedicado de SVXLink:** Activación bajo demanda mediante un código DTMF específico, permitiendo que el controlador nativo gestione las prioridades de transmisión, las identificaciones reglamentarias y el temporizador de corte.
- **Puente de audio virtual:** Enlace entre SVXLink y el servicio de inteligencia artificial mediante el módulo de kernel `snd-aloop` o tuberías con nombre (*named pipes*), disparando respuestas únicamente frente a una palabra clave definida y cediendo siempre prioridad a los operadores humanos en el canal.

Referencia de origen: Demostración operativa de asistente de voz con PTT automático en 145,550 MHz FM (EA3JIE), disponible en `https://youtube.com/shorts/3LvDuzpY72Y`.