Talk to Your Local LLM: Voice Chat Setup

February 1, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** A local voice pipeline chains three pieces: Whisper (speech-to-text) → your LLM → a TTS engine (text-to-speech). The easiest way to start is Open WebUI, which has built-in voice chat with any Ollama model. For the best standalone setup, use faster-whisper (turbo) for STT and Kokoro for TTS — you'll get ~1 second total latency on a mid-range GPU. Total VRAM cost: about 2-4 GB on top of whatever your LLM needs.

📚 **More on this topic:** Open WebUI Setup Guide · Best Models for Chat · Local RAG Guide · VRAM Requirements

Talking to your local LLM instead of typing is one of those things that sounds like a gimmick until you try it. Once you can just speak a question and hear the answer back, it changes how you interact with local AI entirely.

The pipeline is simpler than you'd think: Whisper listens to you, your LLM thinks, and a TTS engine reads the response aloud. Three pieces, all running locally, no cloud required.

Here's how to set it up.

How the Voice Pipeline Works

Every local voice assistant follows the same three-step chain:

1. **STT (Speech-to-Text):** Microphone audio → Whisper → text transcript
2. **LLM (Language Model):** Text prompt → Ollama/llama.cpp → text response
3. **TTS (Text-to-Speech):** Text response → TTS engine → speaker audio

The total latency is the sum of all three. On a decent GPU, you're looking at:

Stage	Typical Latency	What Drives It
STT (Whisper turbo)	100-300 ms	Audio length, model size, GPU speed
LLM (time to first token)	200-500 ms	Model size, context length, GPU speed
TTS (first audio chunk)	100-300 ms	TTS engine, voice quality, GPU/CPU

Stage	Typical Latency	What Drives It
Total round-trip	**500-1100 ms**	Everything above combined

That's roughly 0.5-1.1 seconds from when you stop talking to when you start hearing the answer. Not quite real-time conversation, but fast enough to feel natural.

Step 1: Speech-to-Text with Whisper

OpenAI's Whisper is the de facto standard for local speech recognition. It's open-source, runs on consumer hardware, and is genuinely good — accurate across accents, handles background noise, supports 99 languages.

Whisper Model Sizes

Model	Parameters	VRAM	Relative Speed	Accuracy (WER)
tiny	39M	~1 GB	32x	Rough — fine for commands
base	74M	~1 GB	16x	Decent for clear speech
small	244M	~2 GB	6x	Good general use
medium	769M	~5 GB	2x	Great accuracy
large-v3	1.55B	~10 GB	1x	Best accuracy
turbo	**809M**	**~6 GB**	**6-8x**	**Near large-v3 quality**

The one to use: Whisper turbo. It's 6-8x faster than large-v3 with minimal accuracy loss. Unless you're transcribing heavily accented speech in a noisy room, turbo is the sweet spot.

Which Whisper Implementation?

You have three main options:

faster-whisper (recommended for GPU users): Uses CTranslate2 under the hood. 2-4x faster than OpenAI's original code, uses 40-60% less VRAM. This is what most voice pipeline tools use internally.

```
pip install faster-whisper
```

```
from faster_whisper import WhisperModel

model = WhisperModel("turbo", device="cuda", compute_type="float16")
segments, info = model.transcribe("audio.wav", beam_size=5)

for segment in segments:
    print(segment.text)
```

whisper.cpp (recommended for CPU-only): C/C++ port that runs well without a GPU. Great for laptops and low-power setups. Download GGUF model files and run from the command line:

```
git clone https://github.com/ggerganov/whisper.cpp
cd whisper.cpp && make
./main -m models/ggml-base.en.bin -f audio.wav
```

Original OpenAI Whisper (the baseline): Works but is the slowest option. Only use this if you need a specific feature the others don't support.

```
pip install openai-whisper
whisper audio.wav --model turbo
```

Real-Time Streaming

For voice chat, you don't want to record a whole sentence and then transcribe it. You want streaming — transcribing as you speak. **faster-whisper** supports this with voice activity detection (VAD):

```
from faster_whisper import WhisperModel

model = WhisperModel("turbo", device="cuda")
```

```
# Use Silero VAD to detect speech segments automatically
segments, _ = model.transcribe("audio.wav", vad_filter=True)
```

For a microphone-to-text pipeline, pair faster-whisper with **PyAudio** or **sounddevice** to capture audio chunks, and feed them to the model as they come in.

Step 2: Text-to-Speech Options

TTS is where the local ecosystem has exploded recently. You have options ranging from "sounds like a GPS" to "indistinguishable from a real person."

TTS Comparison

Engine	Size	Runs On	Quality	Latency	Voice Cloning
Kokoro	82M params	CPU or GPU	Excellent — #1 on TTS Arena	Sub-300 ms	No (preset voices)
Piper	15-20M params	CPU only	Good — clear and natural	Sub-100 ms	No (trained voices)
Chatterbox	~300M params	GPU (2-4 GB)	Excellent — beats ElevenLabs in blind tests	Sub-200 ms	Yes (10s reference clip)
Coqui XTTS	~1.5B params	GPU (~2 GB)	Very good, multilingual	300-500 ms	Yes (6s reference clip)
Bark	~1B params	GPU (~12 GB)	Expressive, can laugh/sing	2-5 seconds	Limited
edge-tts	Cloud	Internet required	Very good (Microsoft voices)	100-300 ms	No

Kokoro (Best Overall)

Kokoro is a 82M-parameter model that hit #1 on the TTS Arena leaderboard, beating commercial services. It's fast, sounds natural, and runs on CPU or GPU.

```
pip install kokoro>=0.9
```

```
from kokoro import KPipeline

pipeline = KPipeline(lang_code="a")  # American English
generator = pipeline("Hello! I'm your local AI assistant.", voice="af_heart")

for i, (gs, ps, audio) in enumerate(generator):
    # audio is a numpy array, play or save it
    pass
```

Kokoro has multiple preset voices. No voice cloning, but the built-in voices sound genuinely good.

Piper (Best for CPU-Only)

If you don't have a GPU — or your GPU is fully occupied by the LLM — Piper is the answer. It's a neural TTS engine that runs entirely on CPU with near-instant output:

```
echo "Hello from your local assistant" | \
  piper --model en_US-lessac-medium --output_file response.wav
```

Piper has dozens of pre-trained voices across many languages. Quality is a step below Kokoro or Chatterbox, but speed on CPU is unbeatable.

Download voices from the Piper voices repository.

Chatterbox (Best Voice Cloning)

ResembleAI's Chatterbox can clone any voice from a 10-second audio clip and generates speech that beat ElevenLabs in blind listener tests. It's open-source and runs locally:

```
pip install chatterbox-tts
```

```
import torchaudio
from chatterbox.tts import ChatterboxTTS

model = ChatterboxTTS.from_pretrained(device="cuda")
```

```
text = "Here's your local AI assistant, speaking in a cloned voice."
wav = model.generate(text, audio_prompt_path="reference_voice.wav")
torchaudio.save("output.wav", wav, model.sr)
```

Needs about 2-4 GB VRAM. The voice cloning is genuinely impressive for a local model.

Skip These (For Now)

Bark: Sounds amazing — it can express emotions, laugh, even sing. But it's painfully slow (2-5 seconds per sentence) and needs ~12 GB VRAM. Fun to demo, impractical for conversation.

edge-tts: Great quality and free, but it sends audio through Microsoft's servers. Not local. Fine if you don't care about privacy, but defeats the purpose for most readers of this site.

The Easy Way: Open WebUI Voice Chat

If you just want to talk to your local LLM without building anything, Open WebUI has built-in voice chat. It works with any Ollama model.

Setup

1. Install Ollama and pull a chat model:

```
ollama pull qwen3:8b
```

1. Install Open WebUI:

```
docker run -d -p 3000:8080 \
  --add-host=host.docker.internal:host-gateway \
  -v open-webui:/app/backend/data \
  --name open-webui \
  ghcr.io/open-webui/open-webui:main
```

1. Open `http://localhost:3000`, create an account, select your model.

2. Click the microphone icon in the chat input to use voice. Open WebUI uses your browser's built-in speech recognition (Web Speech API) for STT and can use various TTS backends.

Configuring Better TTS

Open WebUI's default TTS is basic. For better quality, go to **Settings → Audio** and configure:

- **STT Engine:** Set to "whisper (local)" if you want fully local transcription
- **TTS Engine:** Point to a local TTS server (Kokoro or Piper running as a service)

This gives you a ChatGPT-like voice interface running entirely on your machine.

The DIY Way: Command-Line Voice Pipeline

For maximum control, you can wire up the pipeline yourself. Here's a minimal working example using faster-whisper + Ollama + Kokoro:

```
import subprocess
import sounddevice as sd
import numpy as np
from faster_whisper import WhisperModel
from kokoro import KPipeline
import requests
import json

# Initialize models
whisper = WhisperModel("turbo", device="cuda", compute_type="float16")
tts = KPipeline(lang_code="a")

def record_audio(duration=5, sample_rate=16000):
    """Record from microphone."""
    print("Listening...")
    audio = sd.rec(int(duration * sample_rate),
                   samplerate=sample_rate, channels=1, dtype="float32")
    sd.wait()
    return audio.flatten()

def transcribe(audio):
    """Speech to text with faster-whisper."""
    segments, _ = whisper.transcribe(audio, beam_size=5)
    return " ".join(s.text for s in segments).strip()

def query_llm(prompt):
```

```
    """Send text to Ollama and get response."""
    response = requests.post("http://localhost:11434/api/generate",
        json={"model": "qwen3:8b", "prompt": prompt, "stream": False})
    return response.json()["response"]

def speak(text):
    """Text to speech with Kokoro."""
    for _, _, audio in tts(text, voice="af_heart"):
        sd.play(audio, samplerate=24000)
        sd.wait()

# Main loop
while True:
    audio = record_audio()
    text = transcribe(audio)
    if not text:
        continue
    print(f"You: {text}")
    response = query_llm(text)
    print(f"AI: {response}")
    speak(response)
```

This is intentionally simple. For a real setup, you'd add:

- Voice activity detection (VAD) instead of fixed-duration recording
- Streaming LLM output to TTS (speak while generating)
- Conversation history (multi-turn context)
- A wake word or push-to-talk

Dedicated Voice Chat Apps

If you want something more polished than a script but more customizable than Open WebUI:

Moshi (Full-Duplex)

Moshi from Kyutai is the first open-source **full-duplex** voice model — it can listen and talk simultaneously, like a phone call. No STT→LLM→TTS chain; it's a single speech-to-speech model.

- 7B backbone with Mimi audio codec
- 160-200 ms latency
- Runs on a single GPU (needs ~16 GB VRAM)

- Genuinely feels like talking to someone

The catch: it's a fixed model. You can't swap in your favorite LLM. The voice quality and intelligence are whatever Moshi provides. But for natural conversation flow, nothing else comes close locally.

```
pip install moshi
python -m moshi.run
```

RealtimeVoiceChat

An open-source project that wires up the full STT→LLM→TTS pipeline with a web interface and push-to-talk. Supports Ollama, faster-whisper, and multiple TTS engines.

Pipecat

A framework for building voice assistants. More complex to set up, but supports interruption handling, different conversation flows, and multiple backend options. Good if you're building something production-grade.

Hardware Requirements

The big question: can your GPU handle the LLM and the voice pipeline simultaneously?

VRAM Budget

Component	VRAM Needed
Whisper turbo (faster-whisper, FP16)	~6 GB
Whisper turbo (faster-whisper, INT8)	~3 GB
Whisper small (faster-whisper, INT8)	~1 GB
Kokoro TTS	~0.5 GB (or CPU)
Piper TTS	0 GB (CPU only)
Chatterbox TTS	2-4 GB
Your LLM	Depends on model

The math for a 12 GB GPU (RTX 3060):

- Whisper small INT8: 1 GB
- Piper TTS: 0 GB (runs on CPU)
- Remaining for LLM: ~10 GB → Qwen3-8B Q4_K_M fits comfortably

The math for a 24 GB GPU (RTX 3090):

- Whisper turbo INT8: 3 GB
- Kokoro TTS: 0.5 GB (or run on CPU)
- Remaining for LLM: ~19 GB → Qwen3-14B Q6_K or Qwen3-32B Q3_K_M

CPU-only setup (no GPU):

- whisper.cpp with base model: runs fine on any modern CPU
- Piper TTS: CPU-native, sub-100ms latency
- LLM: Qwen3-4B Q4_K_M via llama.cpp (~4 GB RAM)
- Total: workable but slower. Expect 2-4 second round-trips.

The Sharing Problem

Whisper and your LLM can't use the GPU simultaneously by default — one waits for the other. This is fine because the pipeline is sequential anyway. Whisper finishes transcribing before the LLM starts generating.

TTS is where it gets tricky. If your TTS runs on GPU, it competes with the LLM for VRAM. The simplest fix: run TTS on CPU (Piper or Kokoro both handle this well) and keep the GPU for Whisper + LLM.

→ Use our Planning Tool to check exact VRAM for your setup.

Latency Optimization Tips

Want to get below 1 second total? Here's what actually helps:

Use Whisper turbo, not large-v3. The accuracy difference is minimal, the speed difference is 6-8x.

Use INT8 quantization for Whisper. Cuts VRAM roughly in half with negligible accuracy loss:

```
model = WhisperModel("turbo", device="cuda", compute_type="int8")
```

Enable VAD (Voice Activity Detection). Faster-whisper's built-in Silero VAD trims silence before transcription, so Whisper only processes actual speech:

```
segments, _ = model.transcribe(audio, vad_filter=True,
                               vad_parameters=dict(min_silence_duration_ms=500))
```

Stream the LLM output to TTS. Don't wait for the full response. Start speaking the first sentence while the LLM generates the rest. This is the single biggest latency improvement — it turns a 3-second wait into perceived sub-second response.

Run TTS on CPU, STT on GPU. Piper and Kokoro are fast enough on CPU that you don't need GPU acceleration. This leaves all VRAM for Whisper and the LLM.

Use a smaller LLM. A Qwen3-4B model generates its first token in ~100 ms on a decent GPU. A 32B model takes 400+ ms. For voice chat, speed matters more than model intelligence — pick the smallest model that gives acceptable answers.

Recommended Setups

Budget (8 GB VRAM or CPU-only)

- **STT:** whisper.cpp with base model (CPU) or faster-whisper small INT8 (GPU)
- **LLM:** Qwen3-4B Q4_K_M via Ollama
- **TTS:** Piper (CPU)
- **Interface:** Open WebUI or custom script
- **Expected latency:** 1.5-3 seconds

Mid-Range (12 GB VRAM)

- **STT:** faster-whisper turbo INT8 (GPU, ~3 GB)
- **LLM:** Qwen3-8B Q4_K_M via Ollama (~5 GB)
- **TTS:** Kokoro (CPU) or Piper (CPU)

- **Interface:** Open WebUI with local Whisper
- **Expected latency:** 0.8-1.5 seconds

High-End (24 GB VRAM)

- **STT:** faster-whisper turbo FP16 (GPU, ~6 GB)
- **LLM:** Qwen3-14B Q6_K via Ollama (~13 GB)
- **TTS:** Kokoro (CPU) or Chatterbox (GPU if VRAM allows)
- **Interface:** Custom pipeline with streaming
- **Expected latency:** 0.5-1.0 seconds

Common Problems

"Whisper keeps transcribing silence." Enable VAD filtering. Without it, Whisper tries to transcribe background noise and outputs garbage. Use `vad_filter=True` in faster-whisper.

"The TTS voice sounds robotic." Switch from Piper to Kokoro. Or try a higher-quality Piper voice — the "medium" and "high" quality voices sound significantly better than "low."

"There's a long pause before the response starts." You're probably waiting for the full LLM response before sending it to TTS. Stream the output and start speaking the first sentence immediately.

"My GPU runs out of memory." Run TTS on CPU (Piper or Kokoro). Use Whisper small or base instead of turbo. Use a smaller LLM quantization (Q3_K_M instead of Q4_K_M).

"CUDA out of memory when switching between Whisper and LLM." Some frameworks don't release VRAM properly. Use `del model; torch.cuda.empty_cache()` between stages, or keep both models loaded if VRAM allows.

Bottom Line

Local voice chat works today and it's easier to set up than you'd think. The fastest path:

1. Install Ollama + Open WebUI for instant voice chat
2. For better quality, add faster-whisper (turbo) and Kokoro
3. For the most natural experience, try Moshi (if you have 16 GB VRAM)

With a 12 GB GPU, you can run the full pipeline — Whisper turbo INT8, an 8B chat model, and Piper TTS — with about 1 second of latency. That's fast enough for actual conversation.

The technology gap between "local voice assistant" and "cloud voice assistant" is closing fast. A year ago this would have required multiple high-end GPUs. Now it runs on an RTX 3060.

Get notified when we publish new guides.

Subscribe — free, no spam

Source: https://insiderllm.com/guides/voice-chat-local-llms-whisper-tts/

Free guides for running AI locally