EVOKNOW.AI · Open Source AI

# Build Your Own Voice Assistant

*Getting started with Hugging Face Speech-to-Speech*

Speech-to-Speech is Hugging Face's open-source framework for building local voice agents. It chains four swappable stages -- voice detection, speech-to-text, a language model, and text-to-speech -- into a low-latency conversation loop that speaks the same protocol as the OpenAI Realtime API.

This guide walks you from install to a talking assistant: the quick start, running fully offline on your own hardware, picking the right models for your machine, and the flags that matter. No prior voice-AI experience required.

Repository: github.com/huggingface/speech-to-speech
Package: pypi.org/project/speech-to-speech · Licence: Apache 2.0
Published by EVOKNOW.AI · Guide version 1.0

# 1. What is Speech-to-Speech?

Speech-to-Speech is an open-source voice agent framework from Hugging Face. Its own summary is direct: "build local voice agents with open-source models." You speak, it listens, thinks, and speaks back -- and every part of that loop runs on hardware you control, with models you choose.

The appeal is that it removes the usual trade-off. Proprietary voice APIs are easy to start with but send your audio to someone else's servers, cost money per minute, and add network latency. Speech-to-Speech keeps the conversation local while still exposing an OpenAI Realtime-compatible WebSocket API, so existing Realtime clients work against it without modification. It is production-tested, too -- it powers thousands of Reachy Mini robots.

# 2. How the pipeline works

A conversation flows through four stages, each running in its own thread and connected by queues. That isolation is what keeps latency low and makes every stage swappable:

- VAD (voice activity detection) -- Silero VAD v5 works out where your speech starts and stops, and handles turn-taking.
- STT (speech-to-text) -- transcribes what you said, optionally streaming partial transcripts as you talk.
- LLM -- generates the reply, streaming text and any tool calls.
- TTS (text-to-speech) -- synthesises the reply and streams the audio back to the client.

Because the stages are decoupled, you can mix and match: a fast local transcriber with a hosted LLM, or an entirely offline stack. Section 6 covers the options.

# 3. What you need

- Python 3.10 or newer.
- A microphone and speakers. Audio is 16 kHz, int16, mono PCM.
- CPU-only works for small models (roughly 4 GB RAM). A CUDA 12.x+ GPU or Apple Silicon with 8 GB+ is recommended for a smooth, low-latency conversation.
- An API key only if you point the LLM stage at a hosted service -- running fully local needs no keys at all.

# 4. Installing

### From PyPI (recommended)

```
pip install speech-to-speech
```

Optional backends are installed as extras -- add only what you plan to use:

```
pip install "speech-to-speech[kokoro]" # Kokoro TTS
pip install "speech-to-speech[pocket]" # Pocket TTS (voice cloning)
pip install "speech-to-speech[faster-whisper]" # Faster Whisper STT
pip install "speech-to-speech[whisper-mlx]" # Whisper on Apple Silicon
```

### From source

```
git clone https://github.com/huggingface/speech-to-speech.git
cd speech-to-speech
uv sync
```

## 5. Your first conversation

The fastest path starts the realtime server with sensible defaults -- Parakeet TDT for transcription, an OpenAI-compatible LLM, and Qwen3-TTS for the voice:

```
export OPENAI_API_KEY=...
speech-to-speech
```

The server listens on ws://localhost:8765/v1/realtime. In a second terminal, start the local client and begin talking:

```
python scripts/listen_and_play_realtime.py --host 127.0.0.1 --port 8765
```

## 6. Choosing your components

Every stage is selected with a flag, so you can tune the stack to your hardware and language needs.

### Speech-to-text (--stt)

- parakeet-tdt -- the default; fast, covers 25 European languages.
- whisper / faster-whisper -- broad multilingual coverage.
- lightning-whisper-mlx -- optimised for Apple Silicon.
- paraformer -- FunASR-based alternative.

### Language model (--llm_backend)

- responses-api -- the default; talks to any OpenAI-compatible endpoint, hosted or self-hosted (vLLM, llama.cpp).
- chat-completions -- the classic Chat Completions API shape.
- transformers -- runs the model locally on CUDA or CPU.
- mlx-lm -- local inference on Apple Silicon.

### Text-to-speech (--tts)

- qwen3 -- the default; multilingual.
- kokoro -- the compact Kokoro-82M voice.
- pocket -- supports voice cloning.
- chattts / mms -- conversational and very broad language coverage respectively.

# 7. Running fully local

To take the cloud out of the loop entirely, serve the language model yourself with llama.cpp and point the pipeline at it. In the first terminal:

```
llama-server -hf ggml-org/gemma-4-E4B-it-GGUF -np 2 -c 65536 -fa on --swa-full
```

Then in the second, start the voice pipeline against that server -- note the empty API key, since none is needed:

```
speech-to-speech \
--model_name "ggml-org/gemma-4-E4B-it-GGUF" \
--responses_api_base_url "http://127.0.0.1:8080/v1" \
--responses_api_api_key ""
```

### On a Mac

Apple Silicon has a one-flag preset that switches on MPS, the MLX-LM backend, Parakeet TDT and Qwen3-TTS via mlx-audio:

```
speech-to-speech --local_mac_optimal_settings
```

### With Docker

The bundled compose file brings up llama.cpp (Gemma 4) and the socket server together:

```
docker compose up
```

# 8. Modes and useful flags

--mode -- realtime (default), local, socket, or websocket. Use websocket to stream raw 16 kHz PCM: --mode websocket --ws_host 0.0.0.0 --ws_port 8765.

--language -- fix the language (en, zh, ...) or set auto to detect it.

--thresh -- VAD sensitivity; raise it in a noisy room (e.g. --thresh 0.6).

--enable_live_transcription -- stream partial transcripts while the user is still speaking.

--device -- cpu, cuda, or mps.

--model_name -- which LLM to use, local or hosted.

# 9. Driving it from Python

Because the server implements the Realtime protocol, the official OpenAI client can talk to it directly -- just point the base URLs at your own machine:

```
from openai import OpenAI

client = OpenAI(
base_url="http://localhost:8765/v1",
websocket_base_url="ws://localhost:8765/v1",
api_key="not-needed",
)

with client.realtime.connect(model="local") as conn:
conn.send({"type": "session.update", "session": {...}})
for event in conn:
print(event.type)
```

Tool calling and interruption handling are supported, so the assistant can be cut off mid-sentence and can call your functions -- the things that make a voice agent feel real.

## 10. Tips & troubleshooting

- Start with the defaults and only swap one stage at a time -- it makes it obvious which component is costing you latency or accuracy.
- If the assistant talks over you or misses the end of sentences, tune --thresh before changing models.
- On a Mac, reach for --local_mac_optimal_settings first; it picks the Apple-Silicon-native backends for you.
- Offloading the LLM to a llama.cpp or vLLM server usually buys more responsiveness than shrinking the STT or TTS models.
- Match the audio format if you are writing your own client: 16 kHz, int16, mono PCM.

## Resources

- Repository & docs: github.com/huggingface/speech-to-speech
- Package: pypi.org/project/speech-to-speech
- More open-source AI picks: evoknow.ai/tools