ChatGPT

Offline Conversational AI Implementations (Fully Local)

Several open-source projects demonstrate **fully offline voice assistants** that combine local speech recognition, a local LLM for dialogue, and speech synthesis on consumer-grade hardware. Below we detail some promising implementations, each including their GitHub (or code) link, key components, hardware needs, language support, licensing, and notable performance/quality tradeoffs.

VoiceChat2 – Local Voice Chat via WebSockets (GitHub: lhl/voicechat2)

- **Components**: Uses **Whisper.cpp** for speech-to-text (OpenAI Whisper in C/C++), **Llama.cpp** for running a local LLM, and **Coqui TTS** (VITS voice) by default for text-to-speech [1]. It supports alternative TTS backends like Piper and StyleTTS2. The system is modular: you can swap in different STT (e.g. Faster Whisper) or LLM models [2].
- **Hardware & Performance**: Optimized for speed with GPU support. On an AMD Radeon 7900 (RDNA3) GPU, it achieves ~1 second end-to-end response latency using a distilled Whisper model and an 8B Llama variant (quantized) [3]. On an RTX 4090, latency can drop to ~300 ms with faster-whisper and streaming inference [2]. It can run on CPU-only systems as well (via llama.cpp and whisper.cpp), but expect higher latency depending on model size. The WebSocket server design enables running the heavy processes on a PC and accessing it from lighter client devices.
- **Language Support**: Whisper models support dozens of languages for transcription [4] [5]. The default LLM (LLaMA-based instruct model) is English-focused. Coqui TTS offers primarily English voices (e.g. VCTK dataset voice), but Piper TTS has models for many languages. You can configure other language models and voices, though most LLMs (LLaMA-2, etc.) perform best in English.
- **Licensing**: Apache 2.0 license [6]. All components are open-source (Whisper MIT; Llama.cpp MIT; Coqui BSD/MIT). *Note:* The original Meta LLaMA weights require a license (LLaMA-2 is free for commercial use under Meta's terms if conditions met). VoiceChat2's Apache license allows commercial derivative work [6].
- **Setup & Use**: Setup is moderately complex – requires installing the WebSocket server and desired models. It provides a web UI with voice activity detection (VAD) for hands-free use [1]. This project prioritizes **performance and quality**: it streams audio in Opus for efficiency and pipelines STT→LLM→TTS to overlap processing [7]. The trade-off is that it benefits from a powerful GPU to truly shine. For a constrained CPU-only prototype, you might use smaller Whisper/LLM models, accepting slower responses. *VoiceChat2* demonstrates that near real-time voice chat is possible fully offline with careful optimization and powerful hardware.

STTTS Voice Assistant Framework (Hackitude) – Whisper/Vosk + GPT4All/Ollama + Coqui/eSpeak

- **Components**: A configurable **Speech-to-Text → Text-to-Text → Text-to-Speech (STTTS)** pipeline [8]. It supports **Vosk** (Kaldi-based) *or* OpenAI **Whisper** for STT, **GPT4All** *or* **Ollama** (server for LLaMA models) for the LLM, and **eSpeak NG** *or* **Coqui TTS** for speech synthesis [9]. All processing is local; models are downloaded and run offline [9].
- **Hardware & Performance**: Designed to scale **down to Raspberry Pi 4** (4GB). For example, on a Pi 4 it can use Vosk (small English or German model) and eSpeak for minimal resource usage [10] [11]. This yields slower, somewhat robotic interactions, but is functional under ~2 GB RAM [12]. On a desktop PC, one can use Whisper (which benefits from even a modest GPU or can run on CPU for short utterances [13]) and a larger LLM via Ollama (e.g. LLaMA 2 7B or StableLM 7B), plus a neural voice from Coqui TTS for higher quality speech. The framework uses separate processes for STT, LLM, and TTS to utilize multi-core CPUs efficiently [14].
- **Language Support**: Highly flexible. Whisper supports multilingual input; Vosk has models for many languages (you specify the model to download). Coqui TTS offers pretrained voices in dozens of languages (and multilingual models), while eSpeak can synthesize in many languages (with a trade-off in quality). The framework has been demonstrated in English and German locales [10] [15]. The LLM component can utilize any local model; for non-English, one could load a multilingual model (if available) or prompt the English model in another language (with mixed results).
- **Licensing**: The framework code (referred to as **STTTS** by the author) appears to be open-source (no explicit license noted, but it only uses OSS components). Vosk (Apache-2.0), Whisper (MIT), GPT4All (MIT), Ollama (Apache-2.0), Coqui TTS (MPL/MIT), and eSpeak NG (GPL) are all open. For **commercial use**, ensure your chosen LLM model permits it (many GPT4All models are based on LLaMA-2 which is commercially usable under Meta's LLaMA-2 license; older LLaMA-1 based models are research-only). eSpeak NG (GPL) means any integrated code linking it might need to be open-sourced as GPL.
- **Setup & Complexity**: This solution is **highly configurable** via a single YAML/JSON config. It requires installing Python and various extras (for sound I/O, chosen STT/ TTS libraries, etc.) [16] [17]. The author provides examples for different hardware: e.g., on Raspberry Pi using `alsa` audio, Vosk STT, Ollama with a 1.6B StableLM model on CPU, and eSpeak NG TTS [18] [19]. For better quality on a PC, you might configure Whisper and a larger Ollama model with Coqui TTS. **Trade-offs**: This framework can achieve *full offline operation on very low-spec devices* [20], at the cost of response speed and voice naturalness. On more capable machines, swapping in heavier models improves quality (Whisper large, LLaMA 13B, neural TTS) but will need more RAM and possibly a GPU. Overall, STTTS provides a **mix-and-match toolkit** to balance quality vs. efficiency for offline assistants.

ALTS (Async Local Talk-back System) – GitHub: alxpez/alts

- **Components**: **OpenAI Whisper** (via the `openai-whisper` Python package) for STT [21], a local **LLM via LiteLLM/Ollama** for dialogue, and **Coqui TTS** for speech output [22]. By default, ALTS uses **Ollama** with a small **StableLM 1.5B/3B model** ("stablelm2") to keep resource usage low [23]. The TTS uses Coqui's English VITS voice (and requires installing eSpeak NG as a backend for phoneme support) [22].
- **Hardware & Performance**: Aimed at **consumer laptops/desktops** (tested on macOS and Windows) [24]. The default StableLM model (≈3B parameters) is "very tiny and quick" [23] – suitable for CPU inference, but its answers may be relatively simple. You can configure other LLM providers via

LiteLLM (e.g. using a GPT4All model or even OpenAI API if desired) [25]. Whisper's performance will depend on model size – Whisper tiny or base can run real-time on modern CPUs for short utterances, whereas large models need GPU acceleration or more time [13]. Coqui TTS with the default VITS English model can usually run faster than real-time on a CPU (though possibly 1-2 seconds per sentence on slower machines). ALTS runs in a **push-to-talk** style – it listens only while you hold a hotkey, which is efficient and avoids needing constant VAD/wake-word monitoring [26]. This makes it practical on modest hardware.

- **Language Support**: Focus is on English out-of-the-box. Whisper can transcribe other languages, but the default LLM (StableLM) is English-trained. Coqui TTS's default model is English; however, Coqui has models in other languages if one wanted to experiment. You could configure ALTS to use a different Ollama model (for example a fine-tuned LLaMA that speaks another language or an MPT-based model with multilingual data), but that is an advanced use-case.
- **Licensing**: *No explicit license file noted.* The project is described as "100% free" [27]. It uses all open-source tools (Whisper MIT, StableLM (Apache-2.0), Coqui TTS MPL/MIT). Without a stated license, it's safest to treat ALTS as open for personal use. For commercial projects, clarify the licensing with the author or be prepared to replace it with your own implementation using the same open libraries.
- **Setup & Usability**: ALTS is relatively **simple to set up**: clone the repo, install Python requirements, and copy the sample config files [28] [29]. You must install **FFmpeg** (for Whisper audio processing) and eSpeak on your system [30] [22]. Once configured, it runs as a background process awaiting the hotkey press to record audio [26]. This design avoids needing a wake word and ensures privacy (no data leaves the machine) [31]. The trade-off is convenience – you must press a key to talk. **Quality vs. speed**: ALTS's default model will respond faster (being small) but is less fluent than larger LLMs. If better responses are needed, one can point it to a bigger model (e.g. LLaMA-2 7B via Ollama) at cost of higher latency. Overall, ALTS provides a **user-friendly starting point** for a local voice assistant on PC, with all processing offline.

GPT4All Voice Assistant – GitHub: Ai-Austin/GPT4ALL-Voice-Assistant

- **Components**: **OpenAI Whisper** for STT (with modifications to ensure it runs offline entirely) [32], **GPT4All** for the LLM backend (can load any GPT4All-compatible local model), and **eSpeak NG** for TTS output [33]. The assistant runs continuously with a background voice activity detector to pick up when you speak (no internet APIs involved) [34].
- **Hardware & Performance**: Designed for typical desktops. Whisper (small or medium model) will utilize CPU or GPU for transcription – the setup instructions mention patching it so it doesn't try to download models at runtime [32]. GPT4All offers a range of model sizes; for example GPT4All-J (a 4GB GPT-J 6B model) or GPT4All-LLaMa 7B can run on a CPU with ~8–16GB RAM. These models usually generate a few tokens per second on CPU, meaning responses might take a handful of seconds for a sentence-length reply. The assistant uses a **continuous listening** approach with VAD, which is more convenient but heavier than push-to-talk. It's still entirely offline – the VAD is likely implemented by monitoring audio energy or using Whisper in short segments. **Performance trade-off**: smaller models (like 7B) yield faster but simpler responses; larger ones (like 13B) improve coherence but may be too slow on CPU.
- **Language Support**: Primarily English. GPT4All models are mostly English (some limited multilingual ability if the underlying model was, but generally not fluent beyond English). Whisper can transcribe many languages, and eSpeak can output many languages (though one would need to configure it to use the appropriate voice/language code). The project doesn't explicitly mention multi-language

usage. For a non-English assistant, one might use a GPT4All model fine-tuned in that language and set eSpeak's voice accordingly.

- **Licensing**: MIT License [35]. Very permissive for modification and commercial use. Do note that individual model files may have their own licenses (e.g. some GPT4All models based on LLaMA-1 were research-only; however, there are GPT4All models based on LLaMA-2 or Mistral 7B which are commercially usable). Whisper (MIT) and eSpeak NG (GPL) are also open source. If distributing a product, the GPL component (eSpeak) may impose share-alike requirements, so one might swap it out for a different TTS (like MIT-licensed Piper or Coqui voices) for a completely permissive stack.
- **Setup & Notes**: The project provides a YouTube tutorial for installation. Key steps include installing the Python libraries and obtaining a GPT4All model file. One must also install `espeak` system packages (the instructions cover Arch Linux using `espeak` and `python-espeak` AUR packages) [36]. After setup, the assistant runs and continuously listens for speech. The code highlights using a system prompt to improve the model's responses (developers recommend providing a role prompt to guide the AI's behavior) [37]. **Quality considerations**: This assistant is straightforward and privacy-focused [34], but the default TTS (eSpeak) has a robotic sound. Replacing it with a neural TTS would improve user experience (at the cost of extra resources). The **strength** of this project is its simplicity and flexibility (any GPT4All model can be plugged in) and an MIT license that makes it easy to build upon for custom or commercial purposes.

Offline LLM Voice Assistant (Yuva-M) – GitHub: Yuva-M/offline_voice-assistant

- **Components**: A minimal pipeline combining **OpenAI Whisper** for capturing/transcribing speech, **Ollama CLI** to run a local **LLaMA-based model**, and **eSpeak NG** (with an Mbrola voice) for TTS [38] [39]. The LLM used in the example is "Llama 3.2 via Ollama" – referring to a LLaMA2 7B variant served by Ollama (possibly quantized) [38]. A short Python script coordinates these: recording audio for a few seconds, running Whisper STT, feeding the text to the LLM (with a concise-system-prompt to keep answers simple), then using eSpeak NG to speak the response [40].
- **Hardware Requirements**: This is geared toward **low-spec PCs or single-board computers**. Whisper can be run with the small or base model on CPU for short utterances. The LLaMA 7B model, if quantized (e.g. 4-bit), can fit in 4–8 GB RAM. The use of eSpeak ensures the TTS is extremely lightweight (it's instantaneous and uses negligible CPU). The entire pipeline was intended to be usable on a system like a Raspberry Pi 4 or a low-end laptop (with the caveat that response generation will be slow if the CPU is weak – a Pi might take tens of seconds per LLM response unless a very tiny model is used). For better performance, one could run Ollama on a machine with more RAM or a small GPU.
- **Language Support**: The default setup is English (Whisper transcribing English speech, the LLM responding in English, and eSpeak with an English Mbrola female voice). Whisper and eSpeak support many languages (just by choosing a different Whisper model or language setting, and an appropriate eSpeak voice). The LLM would need to be changed for other languages – e.g., one could try a multilingual 7B model or translate queries internally – but by design this example focuses on English conversational ability.
- **Licensing**: Not explicitly stated, but given the simplicity of the repository (a few Python files), it's presumably shared for demonstration. It uses only open-source components: Whisper (MIT), Ollama (Apache-2.0, models depend on license), and eSpeak NG (GPL). For any serious use, one should clarify the license or reimplement the logic (which is straightforward given the small codebase). The

approach itself is viable for commercial use if using a commercially licensed model (e.g. LLaMA-2 or an open 7B model) and if replacing eSpeak or complying with GPL.

- **Summary of Setup & Trade-offs**: This project is basically a **minimal reference implementation**. After installing dependencies (Whisper's Python package, Ollama with a model, eSpeak NG) [41] [42], you run `main.py` to start a loop that records audio and processes it. The **trade-off** here is that using eSpeak yields low audio quality (robotic speech), and the LLM's replies are kept short and simple on purpose [39] to accommodate the small model and slow TTS. However, the system is very resource-friendly and **easy to understand or extend**. One could improve it by swapping in, say, Coqui TTS for a more natural voice (at cost of CPU), or using a larger model on a stronger machine. This implementation shows how even a modest setup can achieve an offline voice assistant pipeline end-to-end [38].

SOLA – Smart Offline-First LLM Assistant (GitHub: FlorSanders/Smart_Offline_LLM_Assistant)

- **Components**: **SOLA** is a comprehensive research project that incorporates multiple options for each stage. It can use Whisper, Vosk, or Coqui STT for speech recognition [43]; it supports text generation with a local LLM (via a module called "LlamaEdge," designed for an 8B LLaMA-family model or even fine-tuned TinyLlama) [44]; and for TTS it offers Piper, Coqui TTS, or Mycroft Mimic3 (which is based on Tacotron2) [45] – all running offline. It also includes a wake-word detector and can integrate optional online tools in a controlled way (thus "offline-first," meaning it prefers local processing and only uses external APIs if explicitly enabled) [46] [47]. The architecture is modular and allows benchmarking different components.
- **Hardware & Performance**: Aimed at flexibility rather than a specific hardware target. It has been tested on systems with **GPUs and on Raspberry Pi** (given Piper and Coqui TTS are optimized for Pi4 in some cases [48]). Using Piper TTS and a smaller ASR (like Vosk) would allow SOLA to run on a Pi-class device. Meanwhile, using Whisper and a larger LLM (LLaMA 8B) would require a PC with sufficient RAM and ideally a GPU for better speed. The project provides benchmarking tools to measure latency of different model combinations [49]. With an 8B quantized model on a decent CPU/GPU, expect responses in a few seconds; with TinyLlama (1.1B) one could get faster albeit simpler responses.
- **Language Support**: Very broad. Whisper and Coqui STT are multilingual; Vosk has specific models per language (English, German, Spanish, etc.). Piper and Mimic3 have pretrained voices for many languages (Piper is explicitly optimized for Pi and supports ~20+ languages with various voices [48]). The default "LlamaEdge" model is focused on English (an 8B LLaMA fine-tune for tool use), but one could load other models for other languages if available. In short, SOLA's STT/TTS can be configured for various languages, making it suitable for a multilingual assistant if paired with an appropriate LLM or translation mechanism.
- **Licensing**: Apache 2.0 [50]. This is a positive for commercial use. All integrated components are open source, but note their licenses: Mycroft Mimic3 has Apache/MIT license, Piper is MIT, Coqui STT/TTS have MPL or MIT, etc. The recommended LLM (LLaMA 8B via LlamaEdge) presumably refers to LLaMA-2 7B or similar, which is permitted for commercial use (if usage guidelines are followed).
- **Setup & Trade-offs**: SOLA is more **complex and research-oriented**. It includes configuration files and even a PDF report on its design [51]. Setting it up involves installing Conda environments and various dependencies per the guide [52]. The advantage is **flexibility**: you can experiment with different pipelines (e.g., compare Whisper vs. Vosk accuracy, or Piper vs. Coqui TTS voice quality) in one framework [53]. For someone looking to deploy a product, SOLA might be overkill, but it provides

valuable insight into the performance of different open-source models and shows that a fully local assistant is feasible with current tools. It was built with privacy in mind (no data leaves the device unless you opt to enable online tool plugins) [46]. The authors even explored fine-tuning smaller LLMs (TinyLlama) to improve tool usage [44]. In summary, SOLA is a **cutting-edge example** of an offline voice assistant platform – ideal for learning and testing, and a solid base if you need a highly customizable solution under a permissive license.

LocalTalk (Whisper + LLaMA-2 + Bark) – (GitHub: vndee/local-talking-llm)

- **Components**: This project (from a 2025 tutorial) combines **OpenAI Whisper** for STT, a **LLaMA-2** conversational model served via Ollama for the LLM, and **Suno's Bark** for text-to-speech [54]. It uses LangChain to manage the conversational context with the LLM and handle prompts. Bark is a neural TTS that produces very natural, expressive speech (including intonation, pauses, and even non-English phrases) compared to traditional TTS.
- **Hardware & Performance**: Bark is the most resource-intensive component – it's effectively a GPT-sized model for speech. Recent updates to Bark have improved efficiency (claimed ~10× speed-up on CPU) [55], but it still may require a powerful CPU (or preferably a GPU) to generate speech without long delays. LLaMA-2 7B or 13B via Ollama likewise runs faster with a GPU, though 7B 4-bit can run on CPU with ~8GB RAM. In practice, a high-end desktop would be ideal for this stack: Whisper transcribing in real-time on one GPU core, LLaMA generating a response in a few seconds, and Bark taking a couple more seconds to synthesize high-quality audio. On CPU-only, expect significantly longer synthesis times (potentially 5–10 seconds of processing for each second of speech, depending on model and optimizations).
- **Language Support**: Whisper and Bark are multilingual (Bark can even sing or output in different languages given the right input). The LLaMA-2 model is predominantly English; you could prompt it in other languages, but it might not be as reliable. Bark will voice whatever text it's given, so one approach for multi-language would be to transcribe non-English speech with Whisper and then either use an LLM that supports that language or translate it before feeding to the English LLM, then use Bark to speak the answer (Bark can speak the translated text in the original language if provided). This is complex; by default the demo focuses on English chatbot behavior.
- **Licensing**: Bark is MIT licensed (now cleared for commercial use) [56]. Whisper is MIT. LLaMA-2 is available for commercial use (with some usage restrictions), though if you fine-tune it or use a community model, check that model's license. The code in the tutorial's GitHub (local-talking-llm) is likely MIT as well (to be confirmed from the repository). Overall, this stack is *potentially commercial-friendly*, as none of the core components force code disclosure.
- **Setup & Considerations**: This implementation shows the **highest audio quality output** among the listed solutions, thanks to Bark. The **trade-off** is that Bark (and to a lesser extent Whisper/LLaMA-2) demands more compute. To prototype this, one would install the Python packages for whisper and Bark [57] [58], install Ollama and download a LLaMA-2 model, and run the integration script. The tutorial author uses Poetry for environment management and provides the full code. If you have a capable machine and want a very natural sounding assistant, this approach is promising. For constrained hardware, though, Bark is likely too slow – you'd revert to a lighter TTS. Thus, **LocalTalk** is a proof-of-concept of a *no-compromise voice assistant* (accurate STT, powerful LLM, human-like TTS), illustrating what's possible entirely offline [54] if you're willing to allocate enough computing resources.

Recommendations and Next Steps

Based on the above analysis, **the optimal solution depends on your hardware constraints and quality requirements**:

- For **severely constrained devices (Raspberry Pi, low-end CPUs)**: a pipeline using **Vosk STT + a small GPT-4-All or TinyLlama model + eSpeak/Piper TTS** would be most practical. The Hackitude STTTS framework is a great starting point, as it has configuration examples for Pi (using Vosk and eSpeak) [18] [12]. This will be limited in voice naturalness and LLM sophistication, but it ensures the system runs within 2–4GB RAM and low CPU. Next steps here: set up Vosk with the appropriate language model, install an Ollama or GPT4All backend with a tiny model (e.g. 1–3B parameters), and use eSpeak NG or a lightweight Piper voice for output. Optimize by reducing audio length and keeping prompts short.

- For **moderate consumer hardware (typical laptop/desktop with 8–16GB RAM, no GPU)**: focus on **Whisper (small or medium) + a 7B quantized LLM (Llama-2 7B GGML via llama.cpp/GPT4All) + an efficient neural TTS like Coqui or Piper**. Projects like **ALTS** and the **GPT4All Voice Assistant** already implement this combination with manageable defaults. You could start by cloning ALTS [59] or GPT4All-Voice (MIT licensed), install the requirements, and swap in your chosen model and voice. Ensure Whisper's model is downloaded for offline use (the openai-whisper package can download the model or use Whisper.cpp for even faster CPU inference). For the LLM, download a 4-bit quantized model (e.g. GPT4All's `ggml-gpt4all-j-v1.3.bin` or a Llama-2 7B GGUF model) – these can run on CPU reasonably. For TTS, Coqui's VITS voice *p364* (en-US) or a fast Piper voice will give decent quality with low latency (typically <1 second per sentence on CPU). Assemble these in a simple loop or use the framework provided by those repos. This route balances quality and speed: Whisper's accuracy is high [4], a 7B LLM can handle basic conversations, and the TTS will be understandable and fairly natural. Expect a few seconds total turnaround for a response. Next steps: follow ALTS's instructions to configure the `.env` and `config.yaml` for your environment (choose the model path, etc.) [29], test the pipeline with the hotkey or trigger of your choice, and iteratively improve the voice or model as needed.

- For **higher-end setups (desktop with a decent GPU or 32GB+ RAM)**: you can achieve **near real-time, high-quality conversations**. In this case, consider using **VoiceChat2** or a similar approach with **streaming**. For example, run Whisper.cpp in streaming mode (transcribing audio chunks on-the-fly), use a larger quantized LLM (Llama-2 13B or an 8B specialized model) with streaming token output, and stream the TTS (Coqui TTS and Piper can generate audio in chunks). VoiceChat2 already implements queueing and interleaving of audio output as tokens come in [7] – using it as-is would leverage these optimizations. Next steps for this scenario: set up VoiceChat2's servers (one for STT, one for the LLM, one for TTS) following its README, and try the default models they recommend for speed [3]. Verify the system on your hardware and then adjust models as needed (e.g. swap to faster-whisper or a different voice). With a powerful GPU, you can even try **Bark TTS** for the ultimate voice naturalness, as shown in the LocalTalk project (install Bark and see if your GPU can handle it). Always monitor memory and latency; you might find a sweet spot by quantizing models or using distillations (e.g. the distilled Whisper model used in VoiceChat2 improves STT speed significantly [3]).

In summary, start with a **proven template** from one of the projects above that closest matches your hardware: for a quick prototype on PC, ALTS or the GPT4All Voice Assistant is convenient; for a research-grade solution, VoiceChat2 or SOLA offers extensive capabilities. Then, **incrementally swap components** in or out (Whisper vs. Vosk, different LLMs, different TTS engines) to meet your quality and speed targets. Every component here is interchangeable as long as you handle the I/O (text in/out).

Finally, to set up a **working prototype** step-by-step:

1. **Install Speech Recognition** – e.g. get Whisper.cpp or the `whisper` Python package. Test transcribing a microphone input to ensure STT works locally. If Whisper is too slow, try Whisper tiny model or Vosk for your language.
2. **Set Up the LLM** – install llama.cpp (or an Ollama server, or GPT4All library) and download a smaller LLM model (start with 7B or smaller). Run a few prompt tests in a console to confirm it can generate answers offline. Include a system prompt defining the assistant's persona for better results [37].
3. **Install TTS** – choose a TTS engine: for quick results, `pyttsx3` with your OS's built-in voices or eSpeak NG can give you an immediate voice. For better quality, install Coqui TTS (`pip install TTS`) and download a pretrained model (e.g. "tts_models/en/vctk/vits" for English VCTK voice [60] or use Piper with a suitable voice file). Test the TTS by synthesizing a sample sentence offline.
4. **Integrate and Iterate** – write a simple script (or modify an example from the repos) that takes microphone audio, runs STT, feeds the text to the LLM, then passes the LLM reply to TTS for playback. Start with synchronous processing (record -> transcribe -> generate text -> synthesize -> play audio). Ensure each part works in sequence. Then iterate to improve responsiveness: e.g. implement a wake-word or hotkey trigger, add rudimentary voice activity detection to auto-stop recording, or stream the TTS while the LLM is still finishing (advanced, but VoiceChat2 shows it's possible [7]).

By following the above steps and leveraging the cited projects for reference, you can **build a fully offline conversational AI assistant** on modest hardware. Each component can be tuned to your needs – for instance, use a larger LLM if responses need to be more "intelligent," or a higher-quality TTS voice for a more pleasant user experience, as long as your hardware can support it. With careful selection of models, even a Raspberry Pi can host a basic voice assistant [20], while a mid-range PC can deliver a surprisingly good chat experience entirely offline. The key is to start small, confirm the pipeline end-to-end, and then swap in better models as your hardware allows. With the open-source ecosystem of Whisper, LLaMA-alikes, and TTS libraries, **you have all the building blocks to create a private, offline Siri/Jarvis of your own** – no internet required!

Sources: The information above is drawn from documentation and reports of the respective projects, as cited inline. Each project's repository contains further setup instructions and examples for replication [28] [41] [32] [6].

---

[1] [2] [3] GitHub - lhl/voicechat2: Local SRT/LLM/TTS Voicechat
https://github.com/lhl/voicechat2

[4] [5] [13] Voice-Activated AI with Whisper + LLaMA: Talk to Your Model | by Konna Giann | Apr, 2025 | Medium
https://medium.com/@kgiannopoulou4033/voice-activated-ai-with-whisper-llama-talk-to-your-model-85449748fda0

[6] [7] [49] voicechat2: Local AI Voice Chat - Hackster.io
https://www.hackster.io/lhl/voicechat2-local-ai-voice-chat-4c48f2

[8] [9] [10] [11] [12] [14] [15] [16] [17] [18] [19] [20] LLM Voice Assistant | hackitude
https://www.hackitu.de/llm_voice_assistant/

[21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [59] [60] GitHub - alxpez/alts: 100% free, local & offline voice assistant with speech recognition
https://github.com/alxpez/alts

[32] [33] [34] [35] [36] [37] GitHub - Ai-Austin/GPT4ALL-Voice-Assistant: This is a 100% offline GPT4ALL Voice Assistant. Completely open source and privacy friendly. Use any language model on GPT4ALL. Background process voice detection. Watch the full YouTube tutorial for setup guide: https://youtu.be/6zAk0KHmiGw
https://github.com/Ai-Austin/GPT4ALL-Voice-Assistant

[38] [39] [40] [41] [42] GitHub - Yuva-M/offline_voice-assistant
https://github.com/Yuva-M/offline_voice-assistant

[43] [44] [45] [46] [47] [50] [51] [52] [53] GitHub - FlorSanders/Smart_Offline_LLM_Assistant: SOLA - Smart Offline-first LLM Assistant
https://github.com/FlorSanders/Smart_Offline_LLM_Assistant

[48] rhasspy/piper: A fast, local neural text to speech system - GitHub
https://github.com/rhasspy/piper

[54] [57] [58] Build your own voice assistant and run it locally: Whisper + Ollama + Bark | by Duy Huynh | Medium
https://medium.com/@vndee.huynh/build-your-own-voice-assistant-and-run-it-locally-whisper-ollama-bark-c80e6f815cba

[55] [56] GitHub - suno-ai/bark: Text-Prompted Generative Audio Model
https://github.com/suno-ai/bark