Rust for embedded devices

EchoKit Server

EchoKit

Star, clone and fork 👍

EchoKit devices: https://github.com/second-state/echokit_box

EchoKit server: https://github.com/second-state/echokit_server

Introduction:
https://opencamp.cn/Rust/camp/S02

Sign up here:
https://opencamp.cn/Rust/camp/S02/register?code=cHsXpIq2vGdaM

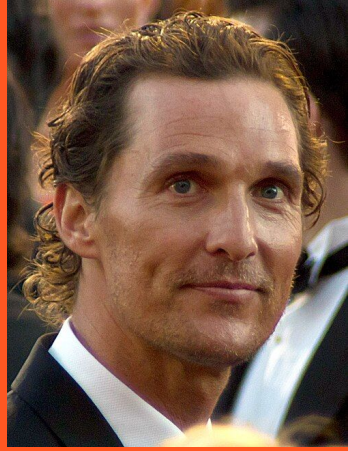
The EchoKit device

An ESP32-S3 SoC + audio processor + microphone + speaker + buttons + USB

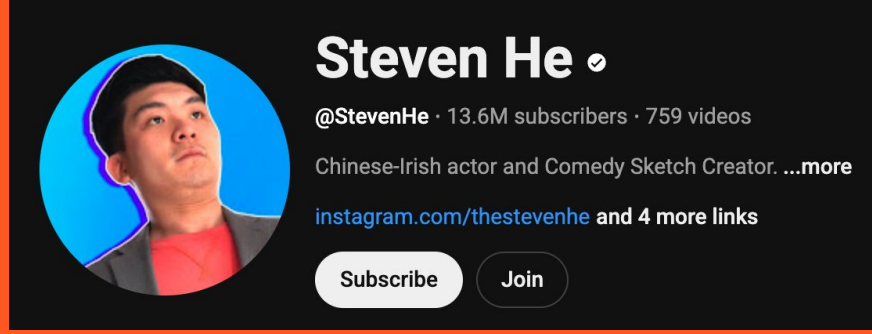
https://opencamp.ai/Rust/bbs/2

How it works

Demo 1: Chat



Demo 2: Texas accent by Matthew McConaughey



Demo 3: Steven He

The basic loop

Workflow

- Detect speech turn on device (VAD) and send to server
- Detect if the speech is complete on server (VAD)
- Transcribe to text (ASR)
- Send text to LLM and get a response (LLM)
- Synthesize response text to speech (TTS)
- Send speech audio data to device

For chat, the entire round trip should be done within 3 seconds.

For real-time and interruptible conversation, it must be under 1 second.

Stage	Time (ms)
macOS mic input	40
Stage	**Time (ms)**
opus encoding	21
network stacks and transit	10
packet handling	2
jitter buffer	40
opus decoding	1
transcription and endpointing	300
llm ttfb	350
sentence aggregation	20
tts ttfb	120
opus encoding	21
packet handling	2
network stacks and transit	10
jitter buffer	40
opus decoding	1
macOS speaker output	15
Total ms	993

ASR: Automatic Speech Recognition

Today there are several very good transcription models that are *not* architected for low latency. Whisper is an open source model that is used in many products and services. It's very good, but usually has a time-to-first-token of 500 ms or more, so is rarely used for conversational voice AI use cases.

4.3.1 Deepgram and Gladia

Most production voice AI agents today use Deepgram [L.07] or Gladia [L.08] for speech-to-text. Deepgram is a commercial speech-to-text AI lab and API platform with a long track record of delivering a very good combination of low latency, low word error rate, and low cost. Gladia is a newer entrant in the space (founded in 2022), with a particular strength in multilingual support.

TTS: Text to Speech

	Cost per min. (approx)	Median TTFB (ms)	P95 TTFB (ms)	avg pre-speech ms
Cartesia	$0.02	190	260	160
Deepgram	$0.008	150	320	260
ElevenLabs Turbo v2	$0.08	300	510	160
ElevenLabs Flash v2	$0.04	170	190	100
Rime	$0.024	340	980	160

What about speech-to-speech?

- Very fast
  - OpenAI realtime API
  - Google Multimedia Live API
- Get more expensive and slower as the conversation progresses
  - Could be as expensive as $7.2/hour – more expensive than humans!
  - Slower than regulate LLM for long conversations
- Very difficult to manage context
  - Cannot do RAG or search
  - No way to do MCP or tool call
- Impossible to finetune
- Cannot clone or customize voice
- No open source solution

Getting started

Build the EchoKit server

```
git clone https://github.com/second-state/echokit_server

cd echokit_server

cargo build --release
```

The config.toml file

```
addr = "0.0.0.0:9090"

hello_wav = "hello.wav"


[[llm.sys_prompts]]

role = "system"

content = """

你是一位乐于助人的助手。请准确、简洁地回答

"""
```

The config.toml file

```
# Get your API key from https://bailian.console.aliyun.com/


[asr]

paraformer_token = "API Key"
```

The config.toml file

```
# Get your API key from https://bailian.console.aliyun.com/

# Any text generation models is supported.


[llm]

llm_chat_url = "https://dashscope.aliyuncs.com/compatible-mode/v1/chat/completions"

api_key = "API-key"

model = "qwen-flash"

history = 5
```

The config.toml file

```
# Get your API key from https://bailian.console.aliyun.com/
# Supported platforms: cosyvoice-v2 (default) and cosyvoice-v1
# Get speaker list from https://help.aliyun.com/zh/model-studio/cosyvoice-java-sdk#722dd7ca66a6x

[tts]
platform = "CosyVoice"
token = "API Key"
speaker = "longhua_v2"
```

Start the EchoKit server

```
nohup target/release/echokit_server &
```

The EchoKit server URL is now at: ws://ip.address:9090/ws/

VAD: Voice Activity Detection

4.7.1 Voice activity detection

Currently, the most common way to do turn detection for voice AI agents is to assume that a long pause means the user has finished speaking.

Voice AI agent pipelines identify pauses using a small, specialized voice activity detection model. A VAD model has been trained to classify audio segments as speech or non-speech. (This is much more robust than trying to identify pauses based only on volume level.)

You can run VAD on either the client-side of a voice AI connection, or on the server. If you need to do significant audio processing on the client anyway, you'll probably need to run VAD on the client to facilitate that. For example, maybe you are identifying wake words on an embedded device, and only sending audio over the network if you detect a wake word at the beginning of a phrase. *Hey, Siri ...*

Generally, though, it's a bit simpler to just run VAD as part of the voice AI agent processing loop. And if your users are connecting via telephone, you don't have a client where you can run VAD, so you have to do it on the server.

The VAD model used most often for voice AI is Silero VAD.[L.26] This open source model runs efficiently on CPU, supports multiple languages, works well for both 8 khz and 16 khz audio, and is available as wasm packages for use in web browsers. Running Silero on a realtime, mono audio stream normally takes less than 1/8th of a typical virtual machine CPU core.

A turn detection algorithm will have a few configuration parameters:

- Length of pause required for end of turn.
- Length of speech segment required to trigger a start speaking event.
- The confidence level for classifying each

On the EchoKit device

Uses an on-device AI model called VADNet.

Trained on 15,000 hours of audio.

https://docs.espressif.com/projects/esp-sr/en/latest/esp32s3/vadnet/README.html

```
afe_config->vad_init = true           // Whether to initial vad in AFE pipeline. Default is true.
afe_config->vad_min_noise_ms = 1000;  // The minimum duration of noise or silence in ms.
afe_config->vad_min_speech_ms = 128;  // The minimum duration of speech in ms.
afe_config->vad_delay_ms = 128;       // The delay between the first frame trigger of VAD and the first frame of speech data.
afe_config->vad_mode = VAD_MODE_1;    // The larger the mode, the higher the speech trigger probability.
```

```rust
unsafe fn afe_init() -> (
    *mut esp_sr::esp_afe_sr_iface_t,
    *mut esp_sr::esp_afe_sr_data_t,
) {
    let models = esp_sr::esp_srmodel_init("model\0".as_ptr() as *const _);
    let afe_config = esp_sr::afe_config_init(
        "M\0".as_ptr() as _,
        models,
        esp_sr::afe_type_t_AFE_TYPE_VC,
        esp_sr::afe_mode_t_AFE_MODE_HIGH_PERF,
    );
    let afe_config = afe_config.as_mut().unwrap();
    afe_config.pcm_config.total_ch_num = 1;
    afe_config.pcm_config.mic_num = 1;
    afe_config.pcm_config.ref_num = 0;
    afe_config.pcm_config.sample_rate = 16000;
    afe_config.afe_ringbuf_size = 25;
    afe_config.vad_min_noise_ms = 500;
    afe_config.vad_mode = esp_sr::vad_mode_t_VAD_MODE_1;
    afe_config.agc_init = true;

    log::info!("{afe_config:?}");

    let afe_ringbuf_size = afe_config.afe_ringbuf_size;
    log::info!("afe ringbuf size: {}", afe_ringbuf_size);

    let afe_handle = esp_sr::esp_afe_handle_from_config(afe_config);
    let afe_handle = afe_handle.as_mut().unwrap();
    let afe_data = (afe_handle.create_from_config.unwrap())(afe_config);
    let audio_chunksize = (afe_handle.get_feed_chunksize.unwrap())(afe_data);
    log::info!("audio chunksize: {}", audio_chunksize);

    esp_sr::afe_config_free(afe_config);
    (afe_handle, afe_data)
}
```

1 init the VAD engine

```rust
_ = async {} => {
    for _ in 0..10{
        let n = rx_driver.read(&mut buf, 100 / PORT_TICK_PERIOD_MS)?;
        afe_handle.feed(&buf[..n]);
    }
    None
}
```

2 Send audio data to the VAD as they come in

```rust
fn afe_worker(afe_handle: Arc<AFE>, tx: MicTx) -> anyhow::Result<()> {
    let mut speech = false;
    loop {
        let result = afe_handle.fetch();
        if let Err(_e) = &result {
            continue;
        }
        let result = result.unwrap();
        if result.data.is_empty() {
            continue;
        }

        if result.speech {
            speech = true;
            log::debug!("Speech detected, sending {} bytes", result.data.len());
            tx.blocking_send(crate::app::Event::MicAudioChunk(result.data))
                .map_err(|_| anyhow::anyhow!("Failed to send data"))?;
            continue;
        }

        if speech {
            log::info!("Speech ended");
            tx.blocking_send(crate::app::Event::MicAudioEnd)
                .map_err(|_| anyhow::anyhow!("Failed to send data"))?;
            speech = false;
        }
    }
}
```

3 Get VAD signal and send to web socket

echokit_box : audio.rs

Build the Silero VAD server

You will need to install libtorch libraries first:
https://github.com/second-state/silero_vad_server

```
git clone https://github.com/second-state/silero_vad_server
cd silero_vad_server
cargo build --release
```

Start the Silero VAD server

```
VAD_LISTEN=0.0.0.0:9093 nohup target/release/silero_vad_server &
```

It starts a web socket service at port 9093: ws://localhost:9093/v1/audio/realtime_vad

4.7.4 Context-aware turn detection (semantic VAD and smart turn)

When humans do turn detection, they use a variety of cues:

- → Identification of filler words like "um" as being likely to indicate continued speech.
- → Grammatical structure.
- → Knowledge of patterns, such as telephone numbers having a specific number of letters.
- → Intonation and pronunciation patterns like drawing out the final word before a pause.

Some recent developments in turn detection:

- → In March, OpenAI shipped a new context-aware turn detection capability for their Realtime API. They call this feature *semantic VAD* , in contrast to the simpler *server VAD* (pause-based turn detection). Docs are here [L.28] .

- → Tavus [L.29] developed a transformer-based native audio turn detection model that is now part of their realtime conversational video API. The Tavus team published a very nice technical overview [L.30] of the problem space and how the model works.
- → The Smart Turn open source model is a state-of-the-art native audio turn detection model built and maintained by the Pipecat community. All training data, training code, inference code, and model weights are open source. [30]

Customized TTS

On device TTS for ESP32

- The standard ESP32 TTS only supports Chinese
  - https://docs.espressif.com/projects/esp-sr/en/latest/esp32s3/speech_synthesis/readme.html
- PicoTTS supports English
  - https://components.espressif.com/components/jmattsson/picotts/
- Problem is that quality is quite poor

Examples

- esp-tts/samples/xiaoxin_speed1.wav (voice=xiaoxin, speed=1): 欢迎使用乐鑫语音合成，支付宝收款 72.1 元，微信收款 643.12 元，扫码收款 5489.54 元
- esp-tts/samples/S2_xiaole_speed2.wav (voice=xiaole, speed=2): 支付宝收款 1111.11 元

Features:

1. **Zero-shot TTS:** Input a 5-second vocal sample and experience instant text-to-speech conversion.
2. **Few-shot TTS:** Fine-tune the model with just 1 minute of training data for improved voice similarity and realism.
3. **Cross-lingual Support:** Inference in languages different from the training dataset, currently supporting English, Japanese, Korean, Cantonese and Chinese.
4. **WebUI Tools:** Integrated tools include voice accompaniment separation, automatic training set segmentation, Chinese ASR, and text labeling, assisting beginners in creating training datasets and GPT/SoVITS models.

https://www.bilibili.com/video/BV12g4y1m7Uw

GPT-SoVITS

The Rust implementation for a TTS server

You will need to install libtorch libraries first:
https://github.com/second-state/gsv_tts

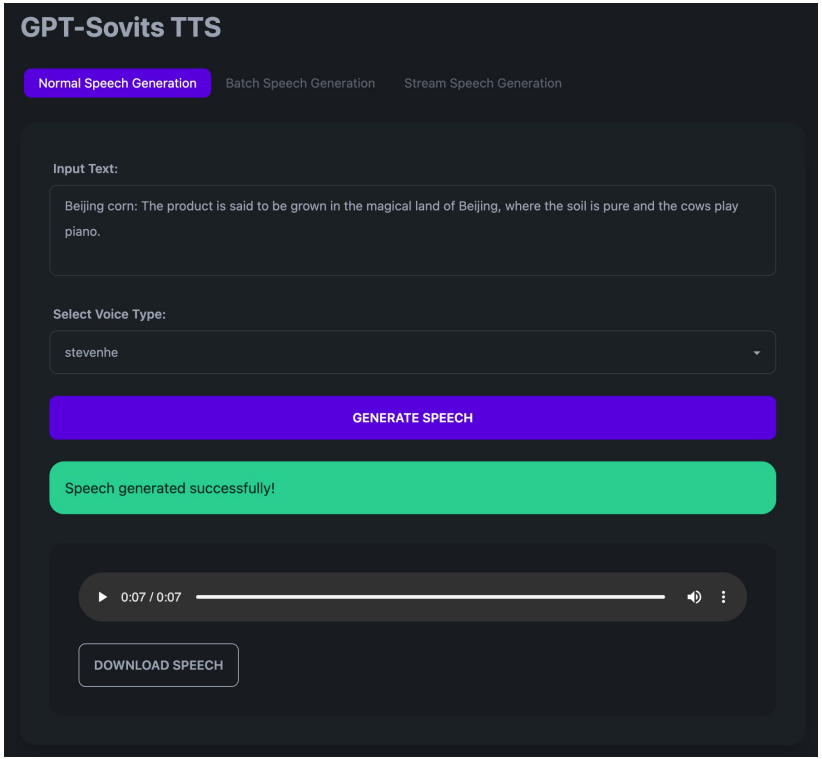
```
git clone https://github.com/second-state/gsv_tts

cd gsv_tts

cargo build --release
```

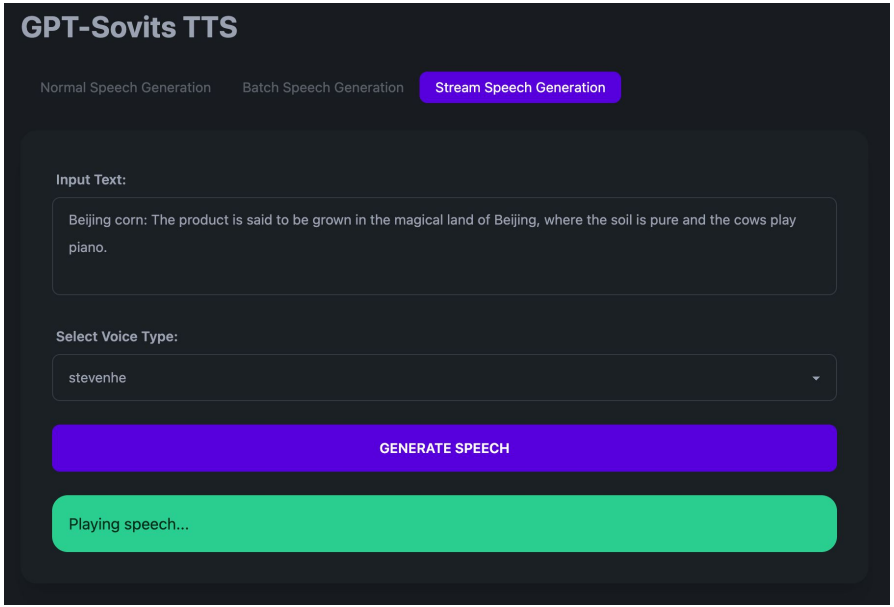
Start the TTS server

```
TTS_LISTEN=0.0.0.0:9094 nohup target/release/gsv_tts &
```

It starts a TTS service at port 9094: http://localhost:9094/

Try ours here: http://35.232.134.140:9094/


GPT-Sovits TTS
Normal Speech Generation
Batch Speech Generation
Stream Speech Generation
Input Text:
Beijing corn: The product is said to be grown in the magical land of Beijing, where the soil is pure and the cows play piano.
Select Voice Type:
stevenhe
GENERATE SPEECH
Speech generated successfully!
0:07 / 0:07
DOWNLOAD SPEECH


GPT-Sovits TTS
Normal Speech Generation
Batch Speech Generation
Stream Speech Generation
Input Text:
Beijing corn: The product is said to be grown in the magical land of Beijing, where the soil is pure and the cows play piano.
Select Voice Type:
stevenhe
GENERATE SPEECH
Playing speech...

Example

The config.toml file

```
[asr]

url = "https://api.openai.com/v1/audio/transcriptions"

api_key = "api_key"

model = "whisper-large-v3"

lang = "auto"

prompt = "Hello\n你好\n(noise)\n(bgm)\n(silence)\n"

vad_realtime_url = "ws://localhost:9093/v1/audio/realtime_vad"
```

The config.toml file

```
[tts]

platform = "StreamGSV"

url = "http://localhost:9094/v1/audio/stream_speech"

speaker = "stevenhe"
```

EchoKit

Star, clone and fork 👍

EchoKit server: https://github.com/second-state/echokit_server

VAD server: https://github.com/second-state/silero_vad_server

TTS server: https://github.com/second-state/gsv_tts

Until next time!