Rust for embedded devices

Self-hosted services

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

EchOKit

Echokit

08/04 16:37:59

嵌入式Rust训练营专用设备

EchoKit

🎓【训练营简介】嵌入式 Rust 训练营是一门面向初学者的项目制学习课程，涵盖嵌入式...

¥168

长按识别小程序 跟团购买

Self-Hosted Servics

The config.toml file (examples/gaia) - general

```
addr = "0.0.0.0:9090"
hello_wav = "hello.wav"

[[llm.sys_prompts]]
role = "system"
content = """
You are a helpful assistant…
If the user is speaking English, you must respond in English.
如果用户说中文，你必须用中文回答。
Si l'utilisateur parle français, vous devez répondre en français.
"""
```

The config.toml file (examples/gaia) - tts

```
# Requires a local gsv_tts server at port 9094: https://github.com/second-state/gsv_tts
[tts]
platform = "StreamGSV"
url = "http://localhost:9094/v1/audio/stream_speech"
speaker = "cooper"
```

The config.toml file (examples/gaia) - asr

```
# Requires a local Whisper API server at port 9092:
https://llamaedge.com/docs/ai-models/speech-to-text/quick-start-whisper
[asr]
url = "http://localhost:9092/v1/audio/transcriptions"
lang = "auto"
# Requires a local Silero VAD server at port 9093:
https://github.com/second-state/silero_vad_server
vad_realtime_url = "ws://localhost:9093/v1/audio/realtime_vad"
```

The config.toml file (examples/gaia) - LLM

```
# Requires a local LlamaEdge API server at port 9091:
https://llamaedge.com/docs/ai-models/llm/quick-start-llm
[llm]
llm_chat_url = "http://localhost:9091/v1/chat/completions"
api_key = "Bearer gaia-1234"
model = "default"
history = 5
```

GVS-TTS

An API server for streaming TTS - libtorch

1. **Install libtorch dependencies**

   Linux x86 CPU

   ```
   curl -LO https://download.pytorch.org/libtorch/cpu/libtorch-shared-with-deps-2.4.0%2Bcpu.zip
   unzip libtorch-shared-with-deps-2.4.0%2Bcpu.zip
   ```

   Linux x86 CUDA

   ```
   curl -LO
   https://download.pytorch.org/libtorch/cu124/libtorch-cxx11-abi-shared-with-deps-2.4.0%2Bcu124.zip
   unzip libtorch-cxx11-abi-shared-with-deps-2.4.0%2Bcu124.zip
   ```

   MacOS on Apple Silicon (M-series) devices

   ```
   curl -LO https://download.pytorch.org/libtorch/cpu/libtorch-macos-arm64-2.4.0.zip
   ```

2. **Then, tell the system where to find your LibTorch.**

   ```
   export LD_LIBRARY_PATH=$(pwd)/libtorch/lib:$LD_LIBRARY_PATH
   export LIBTORCH=$(pwd)/libtorch
   ```

Build the API server

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

cd gsv_tts
cargo build --release
```

Get model files

```
cd resources
curl -LO https://huggingface.co/L-jasmine/GPT_Sovits/resolve/main/v2pro/t2s.pt
curl -LO https://huggingface.co/L-jasmine/GPT_Sovits/resolve/main/v2pro/vits.pt
curl -LO https://huggingface.co/L-jasmine/GPT_Sovits/resolve/main/v2pro/ssl_model.pt
curl -LO https://huggingface.co/L-jasmine/GPT_Sovits/resolve/main/v2pro/bert_model.pt
curl -LO https://huggingface.co/L-jasmine/GPT_Sovits/resolve/main/v2pro/g2pw_model.pt
curl -LO https://huggingface.co/L-jasmine/GPT_Sovits/resolve/main/v2pro/mini-bart-g2p.pt

# If you don't have NVIDIA GPU / CUDA installed, downloading the following models
curl -L -o t2s.pt https://huggingface.co/L-jasmine/GPT_Sovits/resolve/main/v2pro/t2s.cpu.pt
curl -L -o vits.pt
https://huggingface.co/L-jasmine/GPT_Sovits/resolve/main/v2pro/vits.cpu.pt
```

Start the API server

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

`#` **`TTS_LISTEN`** `=> Set the port`

`#` **`nohup`** `=> “no hang up”, keep the server running even the users are logging out`

`#` **`&`** `=> Run the server in the background`

`# Or try ours here: http://35.232.134.140:9094/`

ASR - Whisper

Whisper - Voice-to-Text - Deps

1. **Install wasmedge dependencies**

```
curl -sSf https://raw.githubusercontent.com/WasmEdge/WasmEdge/master/utils/install_v2.sh | bash -s
```

2. **Install whisper plugin**

```
# Download the whisper plugin for Mac Apple Silicon
curl -LO
https://github.com/WasmEdge/WasmEdge/releases/download/0.14.1/WasmEdge-plugin-wasi_nn-whisper-0.14.1-darwin_arm64.tar.gz
tar -xzf WasmEdge-plugin-wasi_nn-whisper-0.14.1-darwin_arm64.tar.gz -C $HOME/.wasmedge/plugin

# Download the whisper plugin for cuda 11.0
curl -LO
https://github.com/WasmEdge/WasmEdge/releases/download/0.14.1/WasmEdge-plugin-wasi_nn-whisper-cuda-11.3-0.14.1-ubuntu20.04_x86_64.tar.gz
tar -xzf WasmEdge-plugin-wasi_nn-whisper-cuda-11.3-0.14.1-ubuntu20.04_x86_64.tar.gz -C $HOME/.wasmedge/plugin

# Download the whisper plugin for cuda 12.0
curl -LO
https://github.com/WasmEdge/WasmEdge/releases/download/0.14.1/WasmEdge-plugin-wasi_nn-whisper-cuda-12.0-0.14.1-ubuntu20.04_x86_64.tar.gz
tar -xzf WasmEdge-plugin-wasi_nn-whisper-cuda-12.0-0.14.1-ubuntu20.04_x86_64.tar.gz -C $HOME/.wasmedge/plugin

# Others, check the release pages:https://github.com/WasmEdge/WasmEdge/releases/tag/0.14.1
```

Download the whisper API server

```
# Download the API server application.
# It's a Wasm file, which is lightweight (the size of the server is 3.7 MB)
# and cross-platform.

curl -LO
https://github.com/LlamaEdge/whisper-api-server/releases/download/0.3.9/whisper-api-server.
wasm

# Or you can build from source by yourself

git clone https://github.com/LlamaEdge/whisper-api-server
cd whisper-api-server
cargo build --release
```

Get model files

```
curl -LO https://huggingface.co/ggerganov/whisper.cpp/resolve/main/ggml-medium.bin

# You can choose any whisper models from here:
# https://huggingface.co/ggerganov/whisper.cpp/tree/main
# From base, tiny, small, medium, large-v1, large-v2, and large-v3
```

Start the API server

```
# Ensure the wasmedge is installed correctly
# You may need to source the ~/.wasmedge/env if the
# binary is fine.

wasmedge --dir .:. whisper-api-server.wasm \
  -m ggml-medium.bin --port 9092

# It will start the whisper API server at port 9092
```

Test the API server

```
# This audio contains a Chinese sentence, 这里是中文广播,
# the English meaning is This is a Chinese broadcast.

curl -LO
https://github.com/LlamaEdge/whisper-api-server/raw/main/data/test_cn.wav

# Sent to API server
curl --location 'http://localhost: 9092/v1/audio/transcriptions' \
  --header 'Content-Type: multipart/form-data' \
  --form 'file=@"test-cn.wav"'

# Expected Output
{
  "text": "[00:00:00.000 --> 00:00:04.000]   这里是中文广播"
}
```

ASR - VAD

An API server for AI VAD - libtorch

1. **Install libtorch dependencies**

   Linux x86 CPU

   ```
   curl -LO https://download.pytorch.org/libtorch/cpu/libtorch-shared-with-deps-2.4.0%2Bcpu.zip
   unzip libtorch-shared-with-deps-2.4.0%2Bcpu.zip
   ```

   Linux x86 CUDA

   ```
   curl -LO
   https://download.pytorch.org/libtorch/cu124/libtorch-cxx11-abi-shared-with-deps-2.4.0%2Bcu124.zip
   unzip libtorch-cxx11-abi-shared-with-deps-2.4.0%2Bcu124.zip
   ```

   MacOS on Apple Silicon (M-series) devices

   ```
   curl -LO https://download.pytorch.org/libtorch/cpu/libtorch-macos-arm64-2.4.0.zip
   ```

2. **Then, tell the system where to find your LibTorch.**

   ```
   export LD_LIBRARY_PATH=$(pwd)/libtorch/lib:$LD_LIBRARY_PATH
   export LIBTORCH=$(pwd)/libtorch
   ```

Build the API server

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

cd silero_vad_server
cargo build --release
```

Run the API server

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

# It starts a websocket service at port 9093:
# ws://localhost:9093/v1/audio/realtime_vad
```

LLM Server

OpenAI compatible API Server

Install wasmedge dependencies

```
    curl -sSf
https://raw.githubusercontent.com/WasmEdge/WasmEdge/master/u
tils/install_v2.sh | bash -s

# If you have multiple plugins installed,
# such as whisper and gguf,
# using WASMEDGE_PLUGIN_PATH=<path/to/plugin/folder> to
# load the corresponding plugins
```

Get the models

```
curl -LO
https://huggingface.co/second-state/Llama-3.2-1B-Instruct-GGUF/resolve/main/Llama-3.2-1B-Instruct-Q5_K_M.gguf

# You can use any GGUF format models,
# including qwen, gpt-oss, and more.
```

Get the API server

```
curl -LO
https://github.com/second-state/LlamaEdge/releases/latest/download/llama-api-server.wasm

# Or build from source
git clone https://github.com/LlamaEdge/LlamaEdge
cd LlamaEdge/llama-api-server
cargo build --release
```

Start the API server

```
# You can also set the API key at the beginning of the execution.
# It's optional.

export LLAMA_API_KEY=<your-api-key>
wasmedge --dir .:. \
  --env API_KEY=$LLAMA_API_KEY \
  --nn-preload default:GGML:AUTO:Llama-3.2-1B-Instruct-Q5_K_M.gguf \
  llama-api-server.wasm \
  -p llama-3-chat \
  --port 9091
```

Interact with the API server - List models

List all models

```
curl -X GET http://localhost:9091/v1/models -H 'accept:application/json'
```

List all models with the API key

```
curl --location 'http://localhost:9091/v1/chat/completions' \
  --header 'Authorization: Bearer <your-api-key>' \
  --header 'Content-Type: application/json'
```

Interact with the API server - Chat

```
# Compose the prompts
curl -X POST http://localhost:9091/v1/chat/completions \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{"messages":[{"role":"system", "content": "You are a helpful
assistant. Try to be as brief as possible."}, {"role":"user", "content":
"Where is the capital of Texas?"}]}'

# Output
{"id":"chatcmpl-5f0b5247-7afc-45f8-bc48-614712396a05","object":"chat.complet
ion","created":1751945744,"model":"Mistral-Small-3.1-24B-Instruct-2503-Q5_K_
M","choices":[{"index":0,"message":{"content":" The capital of Texas is
Austin.","role":"assistant"},"finish_reason":"stop","logprobs":null}],"usage"
:{"prompt_tokens":38,"completion_tokens":8,"total_tokens":46}}
```

Behind the scenes

GVS-TTS

Web Server Entry Point - main.rs

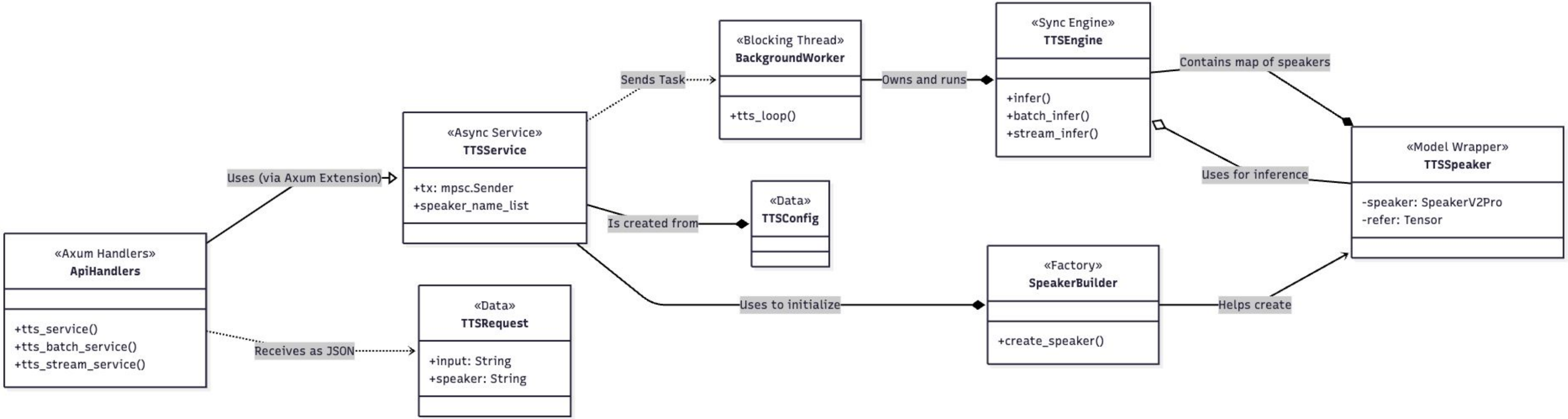
- Using clap to handle the command-line arguments
  - **--listen, --config**
- Loading and deserializing the **config.json** file
- Initializing the core **TTSService**.
- Defining all API routes using the **Axum** router
- Launching the web server to begin listening for HTTP requests.

```
async fn main() {
    let args = Args::parse();
    let config_data = std::fs::read_to_string(args.config).expect(...);
    let tts: TTSConfig = serde_json::from_str(&config_data).expect(...);
    let tts_state = tts::TTSService::create_with_config(tts).expect(...);
    let mut app = app(tts_state);
    let listener = tokio::net::TcpListener::bind(args.listen).await.unwrap();
    axum::serve(listener, app).await.unwrap();
}
```

App Router - main.rs

```
fn app(tts_state: tts::TTSService) -> Router {
    Router::new()
        .route("/v1/audio/speakers", get(tts::tts_speakers_service))
        .route("/v1/audio/stream_speech", post(tts::tts_stream_service))
        .route("/v1/audio/batch_speech", post(tts::tts_batch_service))
        .route("/v1/audio/speech", post(tts::tts_service))
        .fallback(get(index_page))
        .layer(Extension(tts_state))
}
```

The Core TTS Logic - src/tts.rs



Whisper API Server

Arch

[Applications]

--HTTP--------> Hyper Web Server

-------------> Backend Router

(src/backend/[mod.rs](mod.rs))

--dispatch--->Biz Logic Layer

(src/backend/whisper.rs)

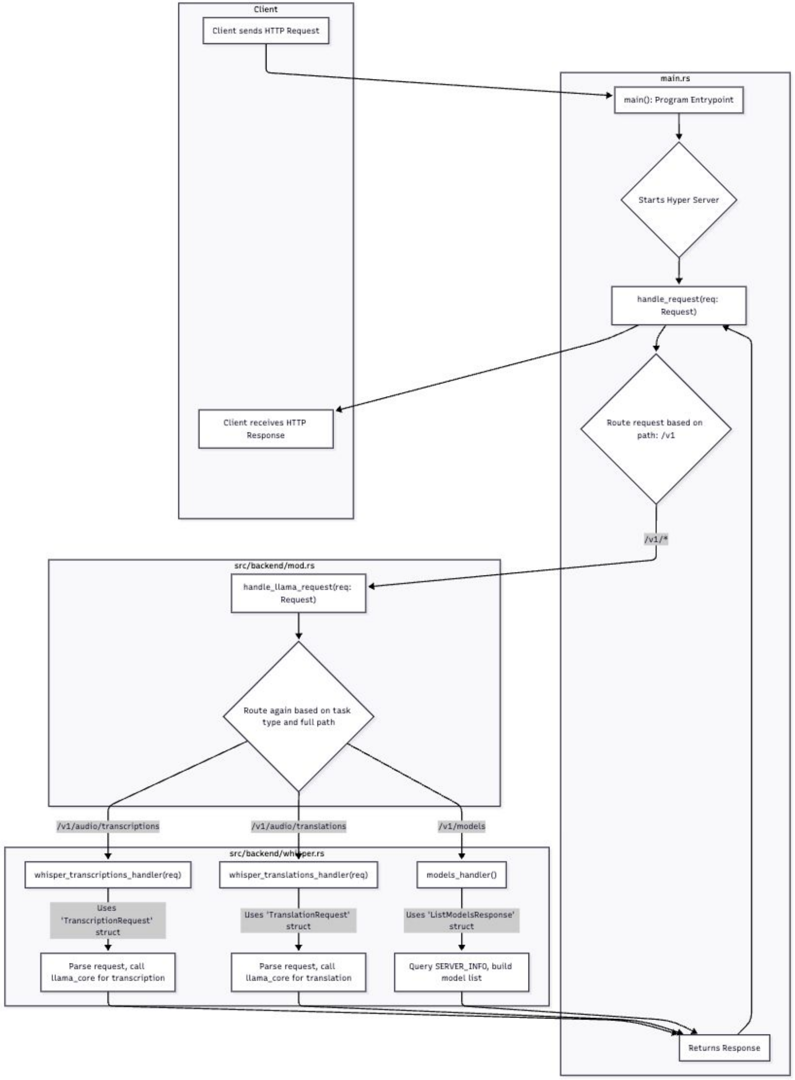
1. audio/transcriptions
2. audio/translations
3. Models
4. Info
5. Files

-------------> llama-core (whisper engine)

1. Load model and initialize
2. Handle the audio and inference
3. Compose the output results

-------------> WasmEdge Runtime

(wasi-nn-whisper plugin) for the execution



File Structure

```
whisper-api-server/
├── src/
│   ├── main.rs            # Entrypoint
│   ├── error.rs           # Errors
│   └── backend/
│       ├── mod.rs         # Router
│       └── whisper.rs     # Biz logic
├── Cargo.toml
├── tests/                 # Test cases
└── data/                  # Testing data
```

LLAMAEDGE API Server

Arch

[Applications]

--HTTP-------> Hyper Web Server

-------------> Backend Router

(src/backend/mod.rs)

--dispatch--->Biz Logic Layer

(src/backend/ggml.rs)

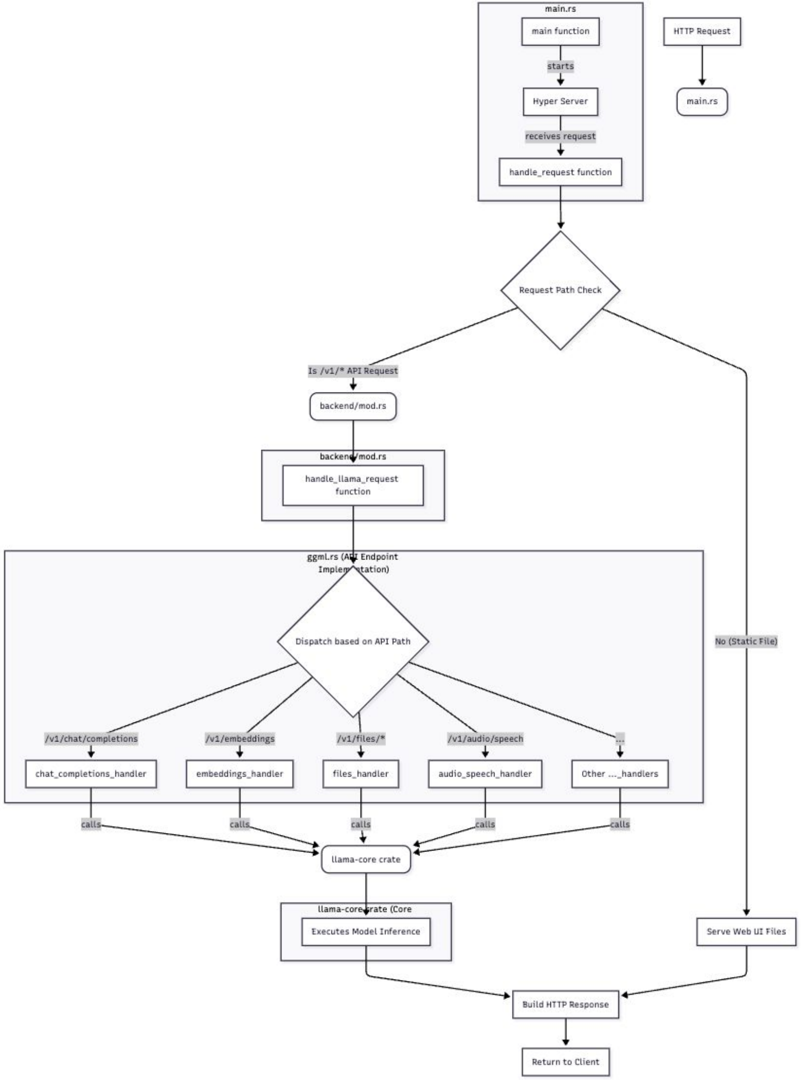
1. **chat/completions**
2. **embedding**
3. Models
4. Info
5. Files

-------------> llama-core (ggml engine)

1. Load model and initialize
2. Handle the input and inference
3. Compose the output results

-------------> WasmEdge Runtime

(wasi-nn-ggml plugin) for the execution



File Structure

```
llama-api-server/
├── src/
│   ├── main.rs              # Entrypoint
│   ├── backend/
│   │   ├── mod.rs           # Router
│   │   └── ggml.rs          # GGML backend
│   ├── config.rs            # config for setting up the server
│   ├── error.rs             # Errors
│   └── utils.rs             # Logs and other utils
├── Cargo.toml
└── README.md
```

SILERO VAD Server

Entrypoint - 1

```
async fn main() {
    let vad_service =
        vad::VadService::new(&args.model_path, 128).expect("Failed to create VAD service");
    let vad_factory = vad::VadFactory::new(args.model_path.clone());
    let app = app(vad_service, Arc::new(vad_factory)); // app router
    let listener = tokio::net::TcpListener::bind(args.listen).await.unwrap();
    axum::serve(listener, app).await.unwrap();
}
```

Entrypoint - 2

```
fn app(vad_service: vad::VadService, vad_factory: Arc<vad::VadFactory>) -> Router {
    Router::new() // Create the app router
        // POST /v1/audio/vad
        // Receive audio, return speeches
        .route("/v1/audio/vad", post(vad::vad_detect))
        // GET /v1/audio/realtime_vad
        // Create WebSocket for the realtime vad
        .route("/v1/audio/realtime_vad", get(vad::websocket_handler))
        // set the body limitation: 10MB
        .layer(DefaultBodyLimit::max(10 * 1024 * 1024))
        .layer(Extension(vad_service))
        .layer(Extension(vad_factory))
}
```

Vad_service - new

```
pub fn new(model_path: &str, buffer: usize) -> anyhow::Result<Self> {
    let vad = silero_vad_jit::VadModelJit::init_jit_model(model_path,
        silero_vad_jit::tch::Device::cuda_if_available(),)?;
    // Use CUDA or CPU
    let mut model = silero_vad_jit::SileroVad::from(vad);
    let _handle = tokio::task:: spawn_blocking(move || {
        while let Some((audio, return_tx)) = rx. blocking_recv() {
            let params = silero_vad_jit::VadParams {
                sampling_rate: 16000,
                ..Default::default()
            };
            match model. get_speech_timestamps(audio, params, None) {
                Ok(timestamps) => {let _ = return_tx.send(Ok(timestamps));}
                Err(e) => {}
            }
        }
    });
    Ok(VadService { tx })
```

Vad_service - detect_vad

```
pub async fn vad_detect(vad_service: Extension<VadService>,mut multipart:
axum::extract::Multipart,) -> impl IntoResponse {
  ...
match vad_service.detect_wav(audio.to_vec()).await {
    Ok(timestamps) => {
        let response = VadResponse {
            timestamps:
timestamps.into_iter().map(SpeechSampleIndex::from).collect(),
        };
        return Json(serde_json::to_value(response).unwrap());
    }
    Err(e) => {
        return Json(serde_json::json!({
            "error": "VAD processing error"
        }));
    }
}
  ...
```

Vad_service - detect_wav

```
pub async fn detect_wav(&self,audio: Vec<u8>,) ->
anyhow::Result<Vec<silero_vad_jit::SpeechTimestamp>> {
    let mut reader = wav_io::reader::Reader::from_vec(audio)?;
    let header = reader.read_header()?;
    let mut samples = reader.get_samples_f32()?;
    // convert stereo to mono
    if header.channels != 1 {
        samples = wav_io::utils:: stereo_to_mono(samples)
    }
    // resample if the sample_rate is not 16kHz
    if header.sample_rate != 16000 {
        samples = wav_io:: resample::linear(samples, 1, header.sample_rate, 16000)
    }
    self.detect_audio_16k(samples).await
}
```

Vad_service - detect_audio_16k

```
pub async fn detect_audio_16k(&self,audio: Vec<f32>,) ->
anyhow::Result<Vec<silero_vad_jit::SpeechTimestamp>> {
    let (return_tx, return_rx) = tokio::sync::oneshot::channel();
    self.tx
        .send((audio, return_tx))
        .await
        .map_err(|_| anyhow::anyhow!("Failed to send audio for VAD processing"))?;

    match return_rx.await {
        Ok(Ok(timestamps)) => Ok(timestamps),
        Ok(Err(e)) => Err(anyhow::anyhow!(e)),
        Err(_) => Err(anyhow::anyhow!("Failed to receive VAD result")),
    }
}
```

SILERO VAD JIT

Usage

```
fn test_vad_basic() {
        let model_path = std::env::var("VAD_MODEL_PATH").unwrap();
        let model = VadModelJit::init_jit_model(&model_path, tch::Device::Cpu).unwrap();
        let mut vad = SileroVad::new(model);

        let audio: Vec<f32> = ...; // audio files in the f32 array
        let params = VadParams {...};
        let progress_callback = ... // progress info
        match vad.get_speech_timestamps(audio, params, progress_callback) {
            Ok(speeches) => {
                println!("detected {} speech segments:", speeches.len());
                for (i, speech) in speeches.iter().enumerate() {
                    println!(...{speech.start,speech.end}...);
                }
            }
            Err(e) => {...}
        }
    }
```

VAD Parameters

```
pub struct VadParams {
    pub threshold: f32,
    pub sampling_rate: usize, // 8kHz or 16kHz
    pub min_speech_duration_ms: usize, // default: 250ms
    pub max_speech_duration_s: f32, // default: f32::INFINITY
    pub min_silence_duration_ms: usize, // default: 100ms
    pub speech_pad_ms: usize, // default: 30ms
    pub return_seconds: bool, // return in seconds or not
    pub visualize_probs: bool, // default: false. Set true if you want the visualization
    pub neg_threshold: Option<f32>, // default: None
}
```

get_speech_timestamps - preprocessing

```
// Check the sampling rate and set the window size for the samples
if sampling_rate != 8000 && sampling_rate != 16000 {}
let window_size_samples = if sampling_rate == 16000 { 512 } else { 256 }; // 16k -> 512, 8k -> 256
self.model.reset_states(); // Reset the model to ensure there are no previous states.

// Convert the duration to samples
let min_speech_samples =
    (sampling_rate as f32 * params.min_speech_duration_ms as f32 / 1000.0) as usize;
let speech_pad_samples =
    (sampling_rate as f32 * params.speech_pad_ms as f32 / 1000.0) as usize;
let max_speech_samples = (sampling_rate as f32 * params.max_speech_duration_s
    - window_size_samples as f32
    - 2.0 * speech_pad_samples as f32) as usize;
let min_silence_samples =
    (sampling_rate as f32 * params.min_silence_duration_ms as f32 / 1000.0) as usize;
let min_silence_samples_at_max_speech = (sampling_rate as f32 * 98.0 / 1000.0) as usize;

let audio_length_samples = audio.len();
```

Step1. Calculate probability

```
let mut speech_probs = Vec::new();
let mut current_start_sample = 0;

while current_start_sample < audio_length_samples {
    // get the current chunk
    let mut chunk = audio[current_start_sample..chunk_end].to_vec();
    if chunk.len() < window_size_samples {
        // if it's the last chunk, adding paddings.
        chunk.resize(window_size_samples, 0.0);
    }
    let speech_prob = self.model.predict(&chunk, sampling_rate)?;
    speech_probs.push(speech_prob);

    // shift to the next chunk
    current_start_sample += window_size_samples;
}
```

Step2. Get speeches from the probability - 1

```
let mut triggered = false; // Check if it’s during a speech
let mut speeches = Vec::new();  // speeches slices
let mut current_speech = SpeechTimestamp::default();  // the current speech
// Check if a speech is ended
let neg_threshold = params
    .neg_threshold
    .unwrap_or_else(|| (params.threshold - 0.15).max(0.01));

// vars for handling the temp information
let mut temp_end = 0;
let mut prev_end = 0;
let mut next_start = 0;
```

Step2. Get speeches from the probability - 2

```
// The main loop, we have four situations need to handled
// This loop will iterate through the probability array.

for (i, speech_prob) in speech_probs.iter().enumerate() {
    let current_sample = window_size_samples * i;
    // Handles cases
}
```

Step2. Get speeches from the probability - 3

```
// Case 1: If a speech is detected, with a temp end position,
//         It’s likely that this speech is a short silence.
if *speech_prob >= params.threshold && temp_end != 0 {
    temp_end = 0;  // clear the temp end position
    if next_start < prev_end {
        next_start = current_sample;
    }
}
```

Step2. Get speeches from the probability - 4

```
// Case 2: If a speech is detected, without the triggered state,
//         this is the start position of the speech
    if *speech_prob >= params.threshold && !triggered {
        triggered = true; // Set it's triggered.
        current_speech.start = current_sample as i64; // record the start pos
        continue;
    }
```

Step2. Get speeches from the probability - 5

```
// Case 3: Speech too long, need to split
if triggered && (current_sample - current_speech.start as usize) >  max_speech_samples
{
    if prev_end > 0 {  // split at the previous end position
        current_speech.end = prev_end as i64;
        speeches.push(current_speech.clone());
        current_speech = SpeechTimestamp::default();
        if next_start < prev_end { triggered = false; }
        else {  current_speech.start = next_start as i64; }
        prev_end = 0; next_start = 0; temp_end = 0; // clean up the positions
    } else {
        current_speech.end = current_sample as i64;
        speeches.push(current_speech.clone());
        current_speech = SpeechTimestamp::default();
        prev_end = 0; next_start = 0; temp_end = 0; triggered = false;
    }
    continue;
}
```

Step2. Get speeches from the probability - 6

```
// Case 4: Detect silence
if *speech_prob < neg_threshold && triggered {
    if temp_end == 0 { temp_end = current_sample; } // record possible end pos
    // if the silence is larger than the min_silience
    // record the possible previous end pos
    if (current_sample - temp_end) > min_silence_samples_at_max_speech {
        prev_end = temp_end;
    }
    // If the length of the silence is less than min_silence, keep waiting
    if (current_sample - temp_end) < min_silence_samples { continue; }
    else { // mark the speech is ended.
        current_speech.end = temp_end as i64;
        if (current_speech.end - current_speech.start) > min_speech_samples as
i64 { speeches.push(current_speech.clone()); }
        // reset the pos info
    }
}
```

Step2. Get speeches from the probability - 7

```
// Handle the last unfinished speech
if current_speech.start > 0
    && (audio_length_samples - current_speech.start as usize) >
min_speech_samples
{
    // The audio has ended; however, the speech is ongoing. Mark it complete.
    current_speech.end = audio_length_samples as i64;
    speeches.push(current_speech);
}
```

Step3. Add paddings

```
for i in 0..speeches.len() {
    // If it’s the first speech, add paddings at the beginning
    // If it’s the last speech, add paddings at the end
    // Otherwise, handle the paddings between the current one and the next one
}
```

Reference

Links

- https://llamaedge.com/docs/ai-models/llm/quick-start-llm/
- https://github.com/second-state/silero_vad_server
- https://github.com/second-state/gsv_tts
- https://github.com/LlamaEdge/LlamaEdge/tree/main/llama-api-server
- https://github.com/LlamaEdge/whisper-api-server
-

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!