ESP32-S3

ESP-SR User Guide

Release master
Espressif Systems
Jan 07, 2026

Table of contents

This document contains ESP-SR usage for ESP32-S3 only.

Chapter 1

Getting Started

Espressif ESP-SR helps you build AI voice solution based on ESP32 or ESP32-S3 chips. This document introduces the algorithms and models in ESP-SR via some simple examples.

1.1 Overview

ESP-SR includes the following modules:

- *Audio Front-end AFE*
- *Wake Word Engine WakeNet*
- *Speech Command Word Recognition MultiNet*
- *Speech Synthesis (only supports Chinese language)*

1.2 What You Need

1.2.1 Hardware

- an audio development board. Recommendation: ESP32-S3-Korvo-1 or ESP32-S3-Korvo-2
- USB 2.0 cable (USB A / micro USB B)
- PC (Linux)

Note: Some development boards currently have the Type C interface. Make sure you use the proper cable to connect the board!

1.2.2 Software

- Download ESP-SKAINET, which also downloads ESP-SR as a component.
- Install the ESP-IDF version recommended in ESP-SKAINET. For detailed steps, please see Section Getting Started in ESP-IDF Programming Guide.

1.3 Compile an Example

- Navigate to ESP-SKAINET/examples/en_speech_commands_recognition .

- Compile and run an example following the instructions.
- The example only supports commands in English. Users can wake up the chip by using wake word "Hi ESP" . Note that the chip stops listening to commands if the users wake up the chip and do not give any commands for some time. In this case, just wake up the chip again by saying the wake word.

Chapter 2

AFE Audio Front-end

2.1 Audio Front-end Framework

2.1.1 Overview

This guide provides an overview of how to use the Audio Front End (AFE) framework and explains the definition of the input format. The AFE framework is designed to process audio data for applications such as speech recognition and voice communication. It includes various algorithms like Acoustic Echo Cancellation (AEC), Noise Suppression (NS), Voice Activity Detection (VAD), and Wake Word Detection (WakeNet).

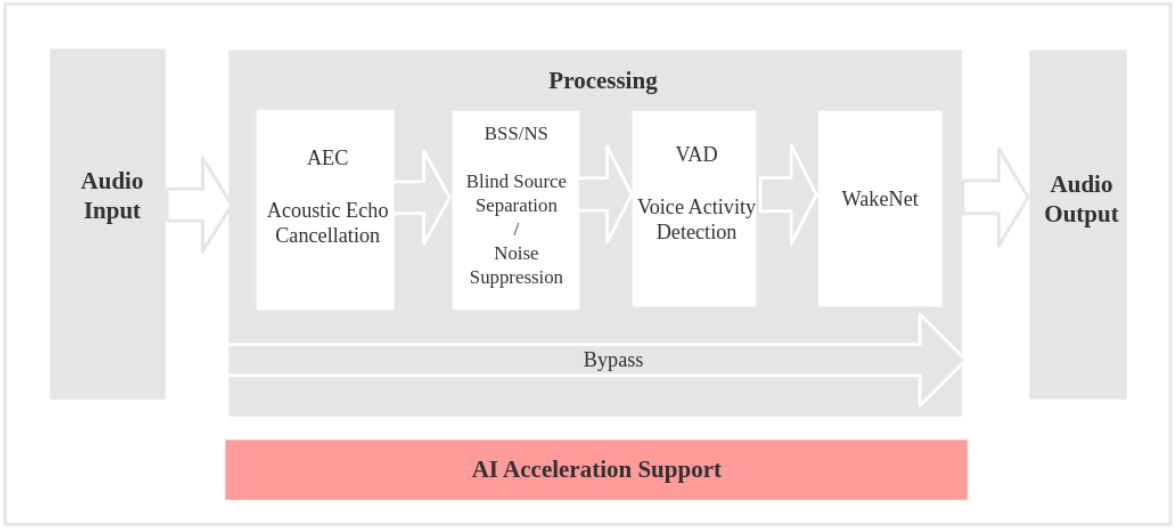
Name	Description
AEC (Acoustic Echo Cancellation)	Supports maximum two-mic processing, which can effectively remove the echo in the mic input signal, and help with further speech recognition.
NS (Noise Suppression)	Supports single-channel processing and can suppress the non-human noise in single-channel audio, especially for stationary noise.
BSS (Blind Source Separation)	Supports dual-channel processing, which can well separate the target sound source from the rest of the interference sound, so as to extract the useful audio signal and ensure the quality of the subsequent speech.
MISO (Multi Input Single Output)	Supports dual channel input and single channel output. It is used to select a channel of audio output with high signal-to-noise ratio when there is no WakeNet enable in the dual mic scene.
VAD (Voice Activity Detection)	Supports real-time output of the voice activity state of the current frame.
AGC (Automatic Gain Control)	Dynamically adjusts the amplitude of the output audio, and amplifies the output amplitude when a weak signal is input; When the input signal reaches a certain strength, the output amplitude will be compressed.
WakeNet	A wake word engine built upon neural network, and is specially designed for low-power embedded MCUs.

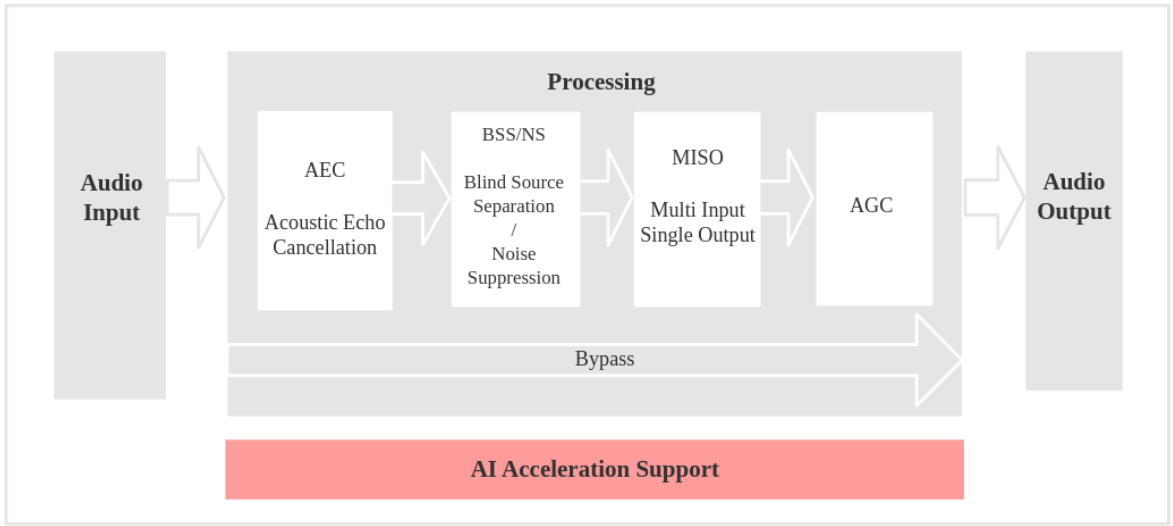
2.1.2 Usage Scenarios

This section introduces two typical usage scenarios of Espressif AFE framework.

Speech Recognition

Voice Communication


Audio Input
Processing
AEC
Acoustic Echo Cancellation
BSS/NS
Blind Source Separation / Noise Suppression
VAD
Voice Activity Detection
WakeNet
Audio Output
Bypass
AI Acceleration Support


Audio Input
Processing
AEC
Acoustic Echo Cancellation
BSS/NS
Blind Source Separation / Noise Suppression
MISO
Multi Input Single Output
AGC
Audio Output
Bypass
AI Acceleration Support

2.1.3 Input Format Definition

The `input_format` parameter specifies the arrangement of audio channels in the input data. Each character in the string represents a channel type:

Character	Description
`M`	Microphone channel
`R`	Playback reference channel
`N`	Unused or unknown channel

Example: `"MMNR"` Indicates four channels inorder : microphone channel, microphone channel, unused channel, and playback reference channel.

Note: The input data must be arranged in **channel-interleaved format**.

2.1.4 Using the AFE Framework

Based on the `menuconfig` -> `ESP Speech Recognition`, select the required AFE (Analog Front End) models, such as the WakeNet model, VAD (Voice Activity Detection) model, NS (Noise Suppression) model, etc., and then call the AFE framework in the code using the following steps.

For reference, you can check the code in test_apps/esp-sr/main/test_afe.cpp or esp-skainet/examples.

Step 1: Initialize AFE Configuration

Get the default configuration using `afe_config_init()` and customize parameters as needed:

```
srmodel_list_t *models = esp_srmodel_init("model");
afe_config_t *afe_config = afe_config_init("MMNR", models, AFE_TYPE_SR, AFE_MODE_
↪HIGH_PERF);
```

- `input_format`: Define the channel arrangement (e.g., `MMNR`).
- `models`: List of models (e.g., for NS, VAD, or WakeNet).
- `afe_type`: Type of AFE (e.g., `AFE_TYPE_SR` for speech recognition).
- `afe_mode`: Performance mode (e.g., `AFE_MODE_HIGH_PERF`).

Step 2: Create AFE Instance

Create an AFE instance using the configuration:

```
// get handle
esp_afe_sr_iface_t *afe_handle = esp_afe_handle_from_config(afe_config);
// create instance
esp_afe_sr_data_t *afe_data = afe_handle->create_from_config(afe_config);
```

Step 3: Feed Audio Data

Input audio data to the AFE for processing. The input data must match the `input_format`:

```
int feed_chunksize = afe_handle->get_feed_chunksize(afe_data);
int feed_nch = afe_handle->get_feed_channel_num(afe_data);
int16_t *feed_buff = (int16_t *) malloc(feed_chunksize * feed_nch * sizeof(int16_
↪t));
afe_handle->feed(afe_data, feed_buff);
```

- `feed_chunksize`: Number of samples to feed per frame.
- `feed_nch`: Number of channel of input data.
- `feed_buff`: Channel-interleaved audio data (16-bit signed, 16 kHz).

Step 4: Fetch Processed Audio

Retrieve the processed single-channel audio data and detection states:

```
afe_fetch_result_t *result = afe_handle->fetch(afe_data);
int16_t *processed_audio = result->data;
vad_state_t vad_state = result->vad_state;
wakenet_state_t wakeup_state = result->wakeup_state;

// if vad cache is exists, please attach the cache to the front of processed_audio␣
↪to avoid data loss
if (result->vad_cache_size > 0) {
    int16_t *vad_cache = result->vad_cache;
}
```

```
// get the processed audio with specified delay, default delay is 2000 ms
afe_fetch_result_t *result = afe_handle->fetch_with_delay(afe_data, 100 / portTICK_
↪PERIOD_MS);
```

2.1.5 Resource Occupancy

For the resource occupancy for AFE, see *Resource Occupancy*.

2.2 Espressif Microphone Design Guidelines

This document provides microphone design guidelines and suggestions for the ESP32-S3 series of audio development boards.

2.2.1 Microphone Electrical Performance Requirement

- Type: omnidirectional MEMS microphone
- Sensitivity
  - Under 1 Pa sound pressure, the sensitivity should be no less than -38 dBV for analog microphones and -26 dB for digital microphones.
  - Tolerance should be within ±2 dB for microphones. And tolerance for microphone arrays should be within ±1 dB.
- Signal-to-noise ratio (SNR)
  - SNR: No less than 62 dB. Higher than 64 dB is recommended.
  - Frequency response should only fluctuate within ±3 dB from 50 to 16 kHz.
  - PSRR should be larger than 55 dB for microphones.

2.2.2 Microphone Structure Design Suggestion

- The aperture or width of the microphone hole is recommended to be greater than 1 mm, the pickup pipe should be as short as possible, and the cavity should be as small as possible. All to ensure that the resonance frequency of the microphone and structural components is above 9 kHz.
- The depth and diameter of the pickup hole are less than 2:1, and the thickness of the shell is recommended to be 1 mm. Increase the hole size of microphone if the shell is too thick.

- The microphone hole must be protected by an anti-dust mesh.
- Silicone sleeve or foam must be added between the microphone and the device shell for sealing and damping, and an interference fit design is required to ensure the leakproofness of the microphone.
- The microphone hole cannot be covered. The microphone in the bottom must keep some clearance from the surfaces such as desktop. Therefore, it's suggested to design some legs for the product to provide such clearance.
- The microphone should be placed far away from the speaker and other objects that can produce noise or vibration, and be isolated and buffered by rubber pads from the speaker sound cavity.

2.2.3 Microphone Array Design Suggestion

Customers can design two or three microphones in an array:

- Two-microphone solution: the distance between the microphones should be 4 ~ 6.5 cm, the axis connecting them should be parallel to the horizontal line, and the center of the two microphones should be horizontally as close as possible to the center of the product.
- Three-microphone solution: the microphones are equally spaced and distributed in a perfect circle with the angle 120 °C from each other, and the spacing should be 4 ~ 6.5 cm.

There are some limitations when selecting microphones for the same array:

- Type: omnidirectional MEMS microphone. Use the same microphone models from the same manufacturer for the array. It's not recommended to use different microphone models in the same array.
- The sensitivity difference of all the microphones in the same array should be within 3 dB.
- The phase difference of all the microphones in the same array should be within 10°.
- It is recommended to use the same structural design for all the microphones in the same array to ensure consistency.

2.2.4 Microphone Leakproofness Suggestion

Use plasticine or similar materials to seal the microphone pickup hole and compare how much the signals collected by the microphone decrease by before and after the seal. 25 dB is qualified, and 30 dB is recommended. Below are the test procedures:

1. Play white noise at 0.5 meters above the microphone, and keep the volume at the microphone 90 dB.
2. Use the microphone array to record for more than 10 s, and store the recording as recording file A.
3. Use plasticine or similar materials to block the microphone pickup hole, record for more than 10 s, and store it as recording file B.
4. Compare the frequency spectrum of the two files and make sure that the overall attenuation in the 100 ~ 8 kHz frequency band is more than 25 dB.

2.2.5 Echo Reference Signal Design Suggestion

- It is recommended that the echo reference signal be as close to the speaker side as possible, and recover from the DA post-stage and PA pre-stage.
- When the speaker volume is at its maximum, the echo reference signal input to the microphone should not have saturation distortion. At the maximum volume, the speaker amplifier output Total Harmonic Distortion (THD) is less than 10% at 100 Hz, less than 6% at 200 Hz, and less than 3% above 350 Hz.
- When the speaker volume is at its maximum, the sound pressure picked up by the microphone does not exceed 102 dB @ 1 kHz.
- The echo reference signal voltage does not exceed the maximum allowed input voltage of the ADC. If it is too high, an attenuation circuit should be added.
- A low-pass filter should be added to introduce the reference echo signal from the output of the Class D power amplifier. The cutoff frequency of the filter is recommended to be more than 22 kHz.
- When the volume is played at the maximum, the recovery signal peak value is -3 to -5 dB.

2.2.6 Microphone Array Consistency Verification

It is required that the difference between the sampled signals of each microphone in the same array is less than 3 dB. Below are the test procedures.

1. Play white noise at 0.5 meters above the microphone, and keep the volume at the microphone 90 dB.
2. Use the microphone array to record for more than 10 s, and check whether the recording amplitude and audio sampling rate of each microphone are consistent.

2.3 Migration from V1.* to V2.*

2.3.1 Input Data Format Changes

The new version use `input_format` string parameter to define the arrangement of audio channels in the input data. Each character in the string represents a channel type:

Character	Description
`M`	Microphone channel
`R`	Playback reference channel
`N`	Unused or unknown channel

Example: `MMNR` indicates four channels, ordered as: microphone channel, microphone channel, unused or unknown channel, playback reference channel.

2.3.2 Configuration and Initialization

- 1. The configuration initialization method AFE_CONFIG_DEFAULT() has been removed. Please use `afe_config_init` to initialize configurations:

```
afe_config_t *afe_config = afe_config_init("MMNR", models, AFE_TYPE_SR,
↪ AFE_MODE_HIGH_PERF);
afe_config_print(afe_config); // print all configurations
```

- 2. ESP_AFE_SR_HANDLE and ESP_AFE_VC_HANDLE have been removed. Use `esp_afe_handle_from_config` to create instances:

```
esp_afe_sr_iface_t *afe_handle = esp_afe_handle_from_config(afe_
↪config);
```

Note: AFE v2.0 introduces additional configuration options. For details, please refer to *AFE* and *VAD*.

Chapter 3

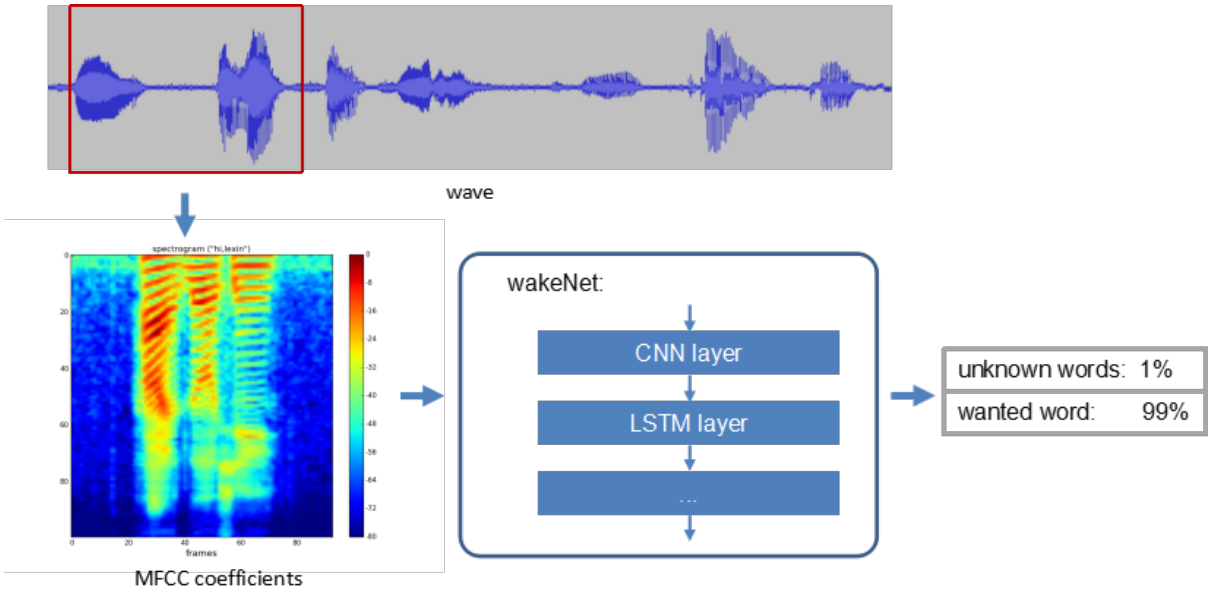
Wake Word

3.1 WakeNet Wake Word Model

WakeNet is a wake word engine built upon neural network for low-power embedded MCUs. Currently, WakeNet supports up to 5 wake words.

3.1.1 Overview

Please see the flow diagram of WakeNet below:



- **Speech Feature** We use MFCC method to extract the speech spectrum features. The input audio file has a sample rate of 16KHz, mono, and is encoded as signed 16-bit. Each frame has a window width and step size of 30ms.
- **Neural Network** Now, the neural network structure has been updated to the ninth edition, among which:
  - WakeNet1, WakeNet2, WakeNet3, WakeNet4, WakeNet5, WakeNet6, and WakeNet7, WakeNet8 had been out of use.
  - WakeNet9 and WakeNet9l support ESP32, ESP32S3, and ESP32P4 chips, which are built upon the Dilated Convolution structure. WakeNet9l further improves the recognition rate of wake words spoken at very fast speeds based on WakeNet9.

  - WakeNet9s supports ESP32C3, ESP32C5 and ESP32C6 chip, which is built upon the Depthwise Separable Convolution structure.

The network structure of WakeNet5, WakeNet5X2 and WakeNet5X3 is the same, but WakeNetX2 and WakeNetX3 have more parameters than WakeNet5. Please refer to *Resource Consumption* for details.

- **Keyword Triggering Method:** For continuous audio stream, we calculate the average recognition results (M) for several frames and generate a smoothing prediction result, to improve the accuracy of keyword triggering. Only when the M value is larger than the set threshold, a triggering command is sent.

3.1.2 Use WakeNet

- Select WakeNet model
  To select WakeNet model, please refer to Section *Flashing Models* .
  To customize wake words, please refer to Section *Espressif Speech Wake-up Solution Customization Process*
- Run WakeNet
  WakeNet is currently included in the *AFE*, which is enabled by default, and returns the detection results through the AFE fetch interface.
  If users do not need WakeNet, please use:

```
afe_config->wakeNet_init = False.
```

  If users want to enable/disable WakeNet temporarily, please use:

```
afe_handle->disable_wakenet(afe_data)
afe_handle->enable_wakenet(afe_data)
```

3.1.3 Resource Occupancy

For the resource occupancy for this model, see *Resource Occupancy*.

3.2 Espressif Speech Wake-up Solution Customization Process

3.2.1 Wake Word Customization Process

Espressif provides users with the **wake word customization** :

1. Espressif has already opened some wake words for customers ' commercial use, such as "HI Leixi" , or "Nihao Xiaoxin" .
   - Espressif also plans to provide more wake words that are free for commercial use soon.
2. Offline wake word customization can also be provided by Espressif:
   - Training corpus provided by customer
     - Customer must provide at least 20,000 qualified corpus entries. See detailed requirements in Section *Requirements on Corpus* .
     - It usually takes two to three weeks for Espressif to train and optimize the received corpus.
     - A fee will be charged for training and optimizing the corpus.
   - Training corpus provided by Espressif
     - Espressif provides all the corpus required for training.
     - The time required to collect corpus needs to be discussed separately. After the corpus is ready, it usually takes two to three weeks for Espressif to train and optimize the received corpus.
     - A fee will be charged for training and optimizing the corpus. A separate fee will be changed for collecting the corpus.
   - The actual fee and time for your customization depend on the **number of wake words you need** and the **scale of your mass production**. For details, please contact our sales person .
3. About Espressif wake word engine WakeNet:
   - Currently, up to 5 wake words are supported by each WakeNet model.
   - A wake word usually consists of 3 to 6 symbols, such as "Hi Leixin" , "xiaoaitongxue" , "nihaotianmao" .
   - More than one WakeNet models can be used together. However, more resource will be consumed when you use more models.
   - For more details, see Section *WakeNet Wake Word Model* .

3.2.2 Requirements on Corpus

As mentioned above, customers can provide Espressif with training corpus collected themselves or purchased from a third party. However, there are some limitations:

- Audio file format
  - Sample rate: 16 kHz
  - Encoding: 16-bit signed int
  - Channel: mono
  - Format: WAV

1. Sampling requirement
   - Number of samples: more than 500 people, including men and women of all ages and at least 100 children.
   - Sampling environment: a quiet room (< 40 dB). It is recommended to use a professional audio room.
   - Recording device: high-fidelity microphone.
   - **How to sample:**
     - At 1 meters away from the microphone: each person speaks the wake word out loud for 15 times (5 times in fast speed, 5 times in normal speed, 5 times in slow speed).
     - At 3 meters away from the microphone: each person speaks the wake word out loud for 15 times (5 times in fast speed, 5 times in normal speed, 5 times in slow speed).
   - File name: it is recommended to name the samples according to the age, gender, and quantity of the collected samples, such as `female_age_fast_id.wav`. Or you can use a separate file to present such information.

3.2.3 Hardware Design and Test

The voice wake-up performance heavily depends on the hardware design and cavity structure. Therefore, please pay special attention to the following requirements:

1. Hardware Design
   - Speaker designs: customers can make their own designs by modifying the reference designs (schematic/PCB) provided by Espressif. Also, Espressif can also review customers' speaker designs to avoid some common design issues.
   - Cavity structure: cavity should be designed by acoustic specialists. Espressif does not provide ID design reference. Customers can refer to other mainstream speaker cavity designs on the market, such as Tmall Genie, Xiaodu Smart Speaker, and Google Smart Speaker, etc.
2. Customers can perform the following tests to verify the hardware designs. Note that it's suggested to perform the following tests in a professional audio room. Customers can adjust the actual tests based on their actual testing environment.
   - Recording test to verify the gain and distortion of mic and codec
     - Play the sample (90 dB, 0.1 meter away from the mic), and adjust the gain to ensure that the recording is not saturated.
     - Use a sweep file of 0~20 KHz, and start recording using the sampling rate of 16 KHz. The recording should not have obvious frequency aliasing.
     - Record 100 samples, and feed these samples to open cloud voice recognition API. A certain recognition rate must be reached.
   - Playback test to verify the distortion of power amplifier (PA) and speaker
     - Test PA power @ 1% Total Harmonic Distortion (THD)
   - Speech algorithms test to verify the AEC, BFM and NS models
     - Adjust the delays of the reference signals based on the different requirements of different AEC algorithms.
     - Test the product based on the actual use scenario. For example, play `85DB_90DB Dreamer.wav` (a song) and record.
     - Analyze the processed signals to evaluate the performance of AEC, BFM, NS, etc.
   - DSP performance test to identify the correct DSP parameter and minimize the nonlinear distortion in the DSP algorithm
     - Noise Suppression
     - Acoustic Echo Cancellation
     - Speech Enhancement
3. Customers can also **send** 1 or 2 pieces of hardware to Espressif and ask us to optimize the product for better wakeup performance.

Chapter 4

Voice Activaty Detection Model

VADNet is a Voice Activaty Detection model built upon neural network for low-power embedded MCUs.

4.1 Overview

VADNet uses a model structure and data processing flow similar to WakeNet, for more details, you can refer to *WakeNet*

VADNet is trained by about 5,000 hours of Chinese data, 5,000 hours of English data, and 5,000 hours of multilingual data.

4.2 Use VADNet

- Select VADNet model

```
idf.py menuconfig
ESP Speech Recognition -> Select voice activity detection -> voice␣
↪activity detection (vadnet1 medium).
```

- Run VADNet

  VADNet is currently included in the *AFE*, which is enabled by default, and returns the detection results through the AFE fetch interface.
  The common vad setting is as follows:

```
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.
```

  If users want to enable/disable/reset VADNet temporarily, please use:

```
afe_handle->disable_vad(afe_data);  // disable VADNet
afe_handle->enable_vad(afe_data);   // enable VADNet
afe_handle->reset_vad(afe_data);    // reset VADNet status
```

- VAD Cache

There are two issues in the VAD settings that can cause a delay in the first frame trigger of speech.

1. The inherent delay of the VAD algorithm itself. VAD cannot accurately trigger speech on the first frame and may delay by 1 to 3 frames.
2. To avoid false triggers, the VAD is triggered when the continuous trigger duration reaches the *vad_min_speech_ms* parameter in AFE configuration.

Due to the above two reasons, directly using the first frame trigger of VAD may cause the first word to be truncated. To avoid this case, AFE V2.0 has added a VAD cache. You can determine whether a VAD cache needs to be saved by checking the vad_cache_size

```
afe_fetch_result_t* result = afe_handle->fetch(afe_data);
if (result->vad_cache_size > 0) {
    printf("vad cache size: %d\n", result->vad_cache_size);
    fwrite(result->vad_cache, 1, result->vad_cache_size, fp);
}

printf("vad state: %s\n", res->vad_state==VAD_SILENCE ? "noise" :
↪"speech");
```

4.3 Resource Occupancy

For the resource occupancy for this model, see *Resource Occupancy*.

Chapter 5

Command Word

5.1 MultiNet Command Word Recognition Model

MultiNet is a lightweight model designed to recognize multiple speech command words offline based on ESP32-S3. Currently, up to 200 speech commands, including customized commands, are supported.

- Support Chinese and English speech commands recognition
- Support user-defined commands
- Support adding / deleting / modifying commands during operation
- Up to 200 commands are supported
- It supports single recognition and continuous recognition
- Lightweight and low resource consumption
- Low delay, within 500ms
- Support online Chinese and English model switching (esp32s3 only)
- The model is partitioned separately to support users to apply OTA

The MultiNet input is the audio processed by the audio-front-end algorithm (AFE), with the format of 16 KHz, 16 bit and mono. By recognizing the audio signals, speech commands can be recognized.

Please refer to *Models Benchmark* to check models supported by Espressif SoCs.

For details on flash models, see Section *Flashing Models* .

Note: Models ending with Q8 represents the 8 bit version of the model, which is more lightweight.

5.2 Commands Recognition Process

Please see the flow diagram for commands recognition below:

5.3 Speech Commands Customization Methods

Note: Mixed Chinese and English is not supported in command words.

The command word cannot contain Arabic numerals and special characters.

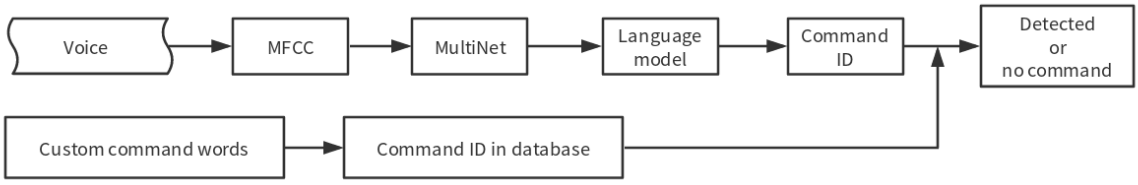


Fig. 1: speech_command-recognition-system

Please refer to Chinese version documentation for Chinese speech commands customization methods.

5.3.1 MultiNet7 customize speech commands

MultiNet7 use phonemes for English speech commands. Please modify a text file model/multinet_model/fst/commands_en.txt by the following format:

```
# command_id,command_grapheme,command_phoneme
1,tell me a joke,TfL Mm c qbK
2,sing a song,Sgl c Sel
```

- Column 1: command ID, it should start from 1 and cannot be set to 0.
- Column 2: command_grapheme, the command sentence. It is recommended to use lowercase letters unless it is an acronym that is meant to be pronounced differently.
- Column 3: command_phoneme, the phoneme sequence of the command which is optional. To fill this column, please use tool/multinet_g2p.py to do the Grapheme-to-Phoneme conversion and paste the results at the third column correspondingly (this is the recommended way).

If Column 3 is left empty, then an internal Grapheme-to-Phoneme tool will be called at runtime. But there might be a little accuracy drop in this way due the different Grapheme-to-Phoneme algorithms used.

5.3.2 MultiNet6 customize speech commands

MultiNet6 use grapheme for English speech commands, you can add/modify speech commands by words directly. Please modify a text file model/multinet_model/fst/commands_en.txt by the following format:

```
# command_id,command_grapheme
1,TELL ME A JOKE
2,MAKE A COFFEE
```

- Column 1: command ID, it should start from 1 and cannot be set to 0.
- Column 2: command_grapheme, the command sentence. It is recommended to use all capital letters.

The extra column in the default *commands_en.txt* is to keep it compatible with MultiNet7, there is no need to fill the third column when using MultiNet6.

5.3.3 MultiNet5 customize speech commands

MultiNet5 use phonemes for English speech commands. For simplicity, we use characters to denote different phonemes. Please use tool/multinet_g2p.py to do the convention.

- Via `menuconfig`
  1. Navigate to `idf.py menuconfig` > `ESP Speech Recognition` > `Add Chinese speech commands/Add English speech commands` to add speech commands. For details, please refer to the example in ESP-Skainet.

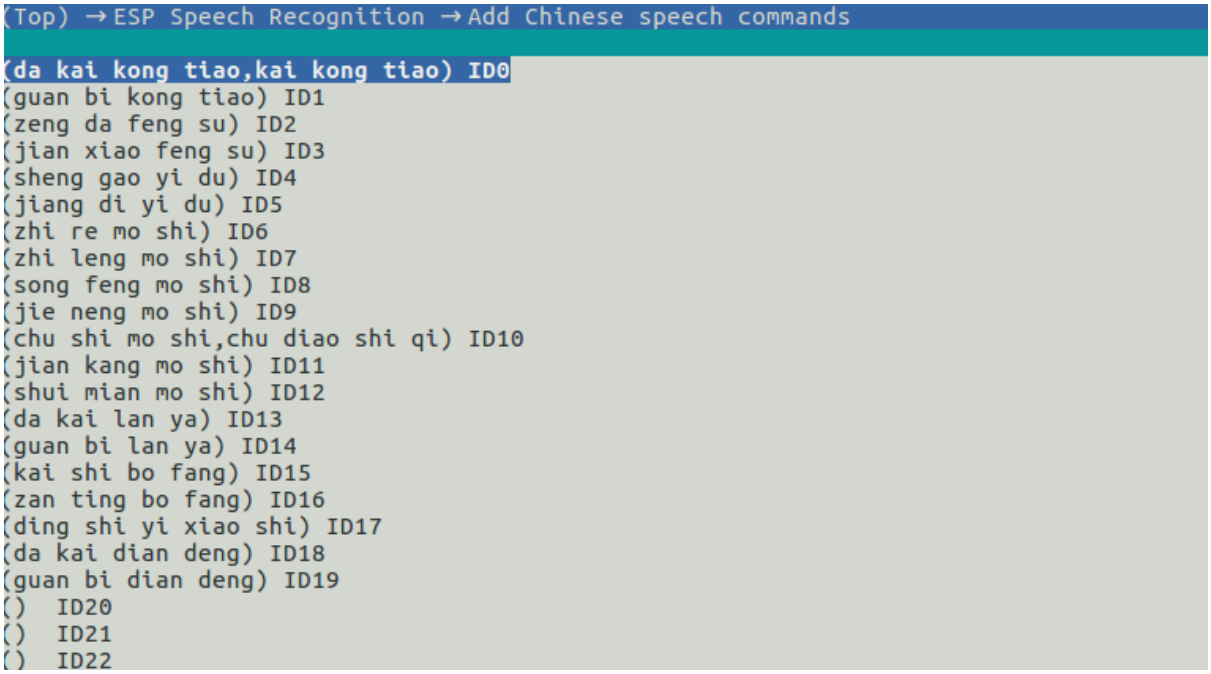


Fig. 2: menuconfig_add_speech_commands

Please note that a single `Command ID` can correspond to more than one commands. For example, "da kai kong tiao" and "kai kong tiao" have the same meaning. Therefore, users can assign the same command id to these two commands and separate them with "," (no space required before and after).

2. Call the following API:

```
/**
    * @brief Update the speech commands of MultiNet by menuconfig
    *
    * @param multinet             The multinet handle
    *
    * @param model_data           The model object to query
    *
    * @param language             The language of MultiNet
    *
    * @return
    *      - ESP_OK                   Success
    *      - ESP_ERR_INVALID_STATE    Fail
    */
    esp_err_t esp_mn_commands_update_from_sdkconfig(esp_mn_iface_t␣
↪*multinet, const model_iface_data_t *model_data);
```

5.3.4 Customize Speech Commands Via API calls

Alternatively, speech commands can be modified via API calls, this method works for MultiNet5, MultiNet6 and MultiNet7.

MutiNet5 requires the input command string to be phonemes, and MultiNet6 and MultiNet7 only accepts grapheme inputs to API calls.

- Apply new changes, the add/remove/modify/clear actions will not take effect util this function is called.

```
/**
* @brief Update the speech commands of MultiNet
```

(continues on next page)

(continued from previous page)

```
*
* @Warning: Must be used after [add/remove/modify/clear] function,
*           otherwise the language model of multinet can not be␣
↪updated.
*
* @return
*     - NULL                 Success
*     - others               The list of error phrase which can not be␣
↪parsed by multinet.
*/
esp_mn_error_t *esp_mn_commands_update();
```

Note: The modifications will not be applied, thus not printed out, until you call `esp_mn_commands_update()`.

- Add a new speech command, will return `ESP_ERR_INVALID_STATE` if the input string is not in the correct format.

```
/**
* @brief Add one speech commands with command string and command ID
*
* @param command_id      The command ID
* @param string  The command string of the speech commands
*
* @return
*     - ESP_OK                  Success
*     - ESP_ERR_INVALID_STATE   Fail
*/
esp_err_t esp_mn_commands_add(int command_id, char *string);
```

- Remove a speech command, will return `ESP_ERR_INVALID_STATE` if the command does not exist.

```
/**
* @brief Remove one speech commands by command string
*
* @param string  The command string of the speech commands
*
* @return
*     - ESP_OK                  Success
*     - ESP_ERR_INVALID_STATE   Fail
*/
esp_err_t esp_mn_commands_remove(char *string);
```

- Modify a speech command, will return `ESP_ERR_INVALID_STATE` if the command does not exist.

```
/**
* @brief Modify one speech commands with new command string
*
* @param old_string  The old command string of the speech commands
* @param new_string  The new command string of the speech commands
*
* @return
*     - ESP_OK                  Success
*     - ESP_ERR_INVALID_STATE   Fail
*/
esp_err_t esp_mn_commands_modify(char *old_string, char *new_string);
```

- Clear all speech commands.

```
/**
* @brief Clear all speech commands in linked list
*
```

(continues on next page)

(continued from previous page)

```
* @return
*      - ESP_OK                  Success
*      - ESP_ERR_INVALID_STATE   Fail
*/
esp_err_t esp_mn_commands_clear(void);
```

- Print cached speech commands, this function will print out all cached speech commands. Cached speech commands will be applied after `esp_mn_commands_update()` is called.

```
/**
* @brief Print all commands in linked list.
*/
void esp_mn_commands_print(void);
```

- Print active speech commands, this function will print out all active speech commands.

```
/**
* @brief Print all commands in linked list.
*/
void esp_mn_active_commands_print(void);
```

5.4 Use MultiNet

We suggest to use MultiNet together with audio front-end (AFE) in ESP-SR. For details, see Section *AFE Introduction and Use* .

After configuring AFE, users can follow the steps below to configure and run MultiNet.

5.4.1 Initialize MultiNet

- Load and initialize MultiNet. For details, see Section *flash_model*
- Customize speech commands. For details, see Section *Speech Commands Customization Methods*

5.4.2 Run MultiNet

Users can start MultiNet after enabling AFE and WakeNet, but must pay attention to the following limitations:

- The frame length of MultiNet must be equal to the AFE fetch frame length
- The audio format supported is 16 KHz, 16 bit, mono. The data obtained by AFE fetch is also in this format
- Get the length of frame that needs to pass to MultiNet

```
int mu_chunksize = multinet->get_samp_chunksize(model_data);
```

  `mu_chunksize` describes the `short` of each frame passed to MultiNet. This size is exactly the same as the number of data points per frame obtained in AFE.
- Start the speech recognition

  We send the data from AFE `fetch` to the following API:

```
esp_mn_state_t mn_state = multinet->detect(model_data, buff);
```

  The length of `buff` is `mu_chunksize * sizeof(int16_t)`.

5.4.3 MultiNet Output

Speech command recognition must be used with WakeNet. After wake-up, MultiNet detection can start.

After running, MultiNet returns the recognition output of the current frame in real time `mn_state`, which is currently divided into the following identification states:

- ESP_MN_STATE_DETECTING
  Indicates that the MultiNet is detecting but the target speech command word has not been recognized.
- ESP_MN_STATE_DETECTED
  Indicates that the target speech command has been recognized. At this time, the user can call `get_results` interface to obtain the recognition results.

```
esp_mn_results_t *mn_result = multinet->get_results(model_data);
```

  The recognition result is stored in the return value of the `get_result` API in the following format:

```
typedef struct{
esp_mn_state_t state;
 int num;                // The number of phrase in list, num<=5. When␣
↪num=0, no phrase is recognized.
 int phrase_id[ESP_MN_RESULT_MAX_NUM];      // The list of phrase id.
 float prob[ESP_MN_RESULT_MAX_NUM];         // The list of probability.
} esp_mn_results_t;
```

  where,
  - `state` is the recognition status of the current frame
  - `num` means the number of recognized commands, `num` <= 5, up to 5 possible results are returned
  - `phrase_id` means the Phrase ID of speech commands
  - `prob` means the recognition probability of the recognized entries, which is arranged from large to small

  Users can use `phrase_id[0]` and `prob[0]` get the recognition result with the highest probability.
- ESP_MN_STATE_TIMEOUT
  Indicates the speech commands has not been detected for a long time and will exit automatically and wait to be waked up again.

Single recognition mode and Continuous recognition mode: * Single recognition mode: exit the speech recognition when the return status is `ESP_MN_STATE_DETECTED` * Continuous recognition mode: exit the speech recognition when the return status is `ESP_MN_STATE_TIMEOUT`

5.5 Resource Occupancy

For the resource occupancy for this model, see *Resource Occupancy*.

Chapter 6

TTS Speech Synthesis Model

Espressif TTS speech synthesis model is a lightweight speech synthesis system designed for embedded systems, with the following main features:

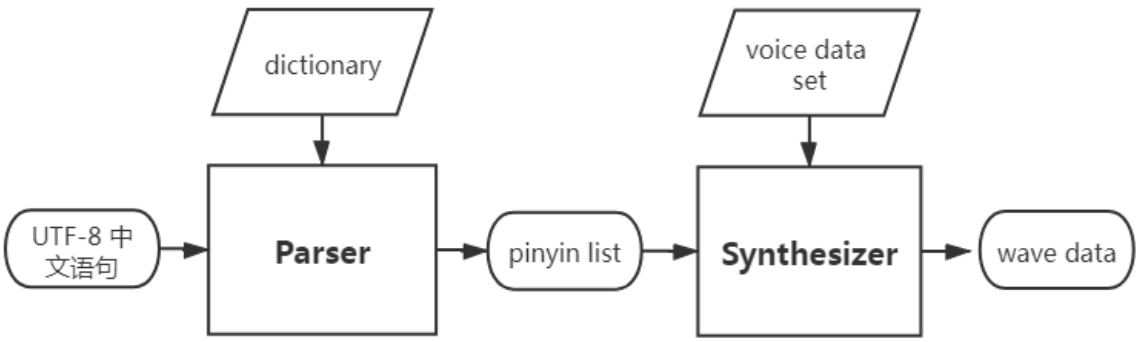
- Currently **Only supports Chinese language**
- Input text is encoded in UTF-8
- Streaming output, which reduces latency
- Polyphonic pronunciation
- Adjustable output speech rate
- Digital broadcasting optimization
- Customized sound set (coming soon)

6.1 Overview

Using a concatenative method, the current version of TTS includes the following components:

- Parser: converts Chinese text (encoded in UTF-8) to phonemes.
- Synthesizer: generates wave raw data from the phonemes provided by the parser and the sound set. Default output format: mono, 16 bit @ 16000 Hz.

Workflow:



6.2 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

6.3 Programming Procedures

```
#include "esp_tts.h"
#include "esp_tts_voice_female.h"
#include "esp_partition.h"

/*** 1. create esp tts handle  ***/


// initial voice set from separate voice data partition

const esp_partition_t* part=esp_partition_find_first(ESP_PARTITION_TYPE_DATA, ESP_
↪PARTITION_SUBTYPE_DATA_FAT, "voice_data");
if (part==0) printf("Couldn't find voice data partition!\n");
spi_flash_mmap_handle_t mmap;
uint16_t* voicedata;
esp_err_t err=esp_partition_mmap(part, 0, part->size, SPI_FLASH_MMAP_DATA, (const␣
↪void**)&voicedata, &mmap);
esp_tts_voice_t *voice=esp_tts_voice_set_init(&esp_tts_voice_template, voicedata);

// 2. parse text and synthesis wave data
char *text="欢迎使用乐鑫语音合成";
if (esp_tts_parse_chinese(tts_handle, text)) {  // parse text into pinyin list
    int len[1]={0};
    do {
        short *data=esp_tts_stream_play(tts_handle, len, 4); // streaming synthesis
        i2s_audio_play(data, len[0]*2, portMAX_DELAY);  // i2s output
    } while(len[0]>0);
    i2s_zero_dma_buffer(0);
}
```

See esp-tts/esp_tts_chinese/include/esp_tts.h for API reference and see the chinese_tts example in ESP-Skainet.

6.4 Resource Occupancy

For the resource occupancy for this model, see *Resource Occupancy*.

Chapter 7

Model Selection and Loading

This document explains how to select and load models for ESP-SR.

7.1 Model Selection

ESP-SR allows you to choose required models through the `menuconfig` interface. To configure models:

1. Run `idf.py menuconfig`
2. Navigate to **ESP Speech Recognition**
3. Configure the following options: - **Noise Suppression Model** - **VAD Model** - **WakeNet Model** - **MultiNet Model**

```
    model data path (Read model data from flash)  --->
    Afe interface (afe interface(version: v1))  --->
    Select noise suppression model (noise suppression (WebRTC))  --->
    Select voice activity detection (voice activity detection (WebRTC))  --->
    Load Multiple Wake Words  --->
    Chinese Speech Commands Model (None)  --->
    English Speech Commands Model (None)  --->
```

7.2 Updating Partition Table

You must add a *partition.csv* file and ensure that there is enough space for the selected models. Add the following line to your project's `partitions.csv` file to allocate space for models:

```
model,  data,        ,         ,     6000K
```

- Replace `6000K` with your custom partition size according to the selected models.
- `model` is the partition label (fixed value).

7.3 Model Loading

7.3.1 ESP-IDF Framework

ESP-SR automatically handles model loading through its CMake scripts:

1. Flash the device with all components: `idf.py flash` *This command automatically loads the selected models.*
2. For code debugging (without re-flashing models): `idf.py app-flash`

Note: The model loading script is defined in `esp-sr/CMakeLists.txt`. Models are flashed to the partition labeled `model` during initial flashing.

7.3.2 Arduino Framework

To manually generate and load models:

1. Use the provided Python script to generate `srmodels.bin`:

```
python {esp-sr_path}/movemodel.py -d1 {sdkconfig_path} -d2 {esp-sr_path} -d3
↪{build_path}
```

Parameters:

- `esp-sr_path`: Path to your ESP-SR component directory
- `sdkconfig_path`: Project's `sdkconfig` file path
- `build_path`: Project's build directory (typically `your_project_path/build`)

2. The generated `srmodels.bin` will be located at: `{build_path}/srmodels/srmodels.bin`
3. Flash the generated binary to your device.

Important: Just regenerate `srmodels.bin` after changing model configurations in `menuconfig`.

Chapter 8

Benchmark

8.1 AFE

8.1.1 Resource Consumption

Table 1: AFE configuration and pipeline

Config	Pipeline
MR, SR, LOW_COST	`|AEC(SR_LOW_COST)| -> |VAD(vadnet1_medium)| -> |WakeNet(wn9_hilexin,)|`
MR, SR, HIGH_PERF	`|AEC(SR_HIGH_PERF)| -> |VAD(vadnet1_medium)| -> |WakeNet(wn9_hilexin,)|`
MR, VC, LOW_COST	`|AEC(VOIP_LOW_COST)| -> |NS(nsnet2)| -> |VAD(vadnet1_medium)|`
MR, VC, HIGH_PERF	`|AEC(VOIP_HIGH_PERF)| -> |NS(nsnet2)| -> |VAD(vadnet1_medium)|`
MMNR, SR, LOW_COST	`|AEC(SR_LOW_COST)| -> |SE(BSS)| -> |VAD(vadnet1_medium)| -> |WakeNet(wn9_hilexin,)|`
MMNR, SR, HIGH_PERF	`|AEC(SR_HIGH_PERF)| -> |SE(BSS)| -> |VAD(vadnet1_medium)| -> |WakeNet(wn9_hilexin,)|`

Note:

- **MR:** one microphone channel and one playback channel
- **MMNR:** two microphone channels and one playback channels
- **Models:** nsnet2, vadnet1_medium, wn9_hilexin

Table 2: AFE configuration and Performance

Config	Internal RAM (KB)	PSRAM (KB)	Feed CPU usage (1 core,%)	Fetch CPU usage (1 core,%)
MR, SR, LOW_COST	72.3	732.7	8.4	15.0
MR, SR, HIGH_PERF	78.0	734.7	9.4	14.9
MR, VC, LOW_COST	50.3	821.4	60.0	8.2
MR, VC, HIGH_PERF	93.7	824.0	64.0	8.2
MMNR, SR, LOW_COST	76.6	1173.9	36.6	30.0
MMNR, SR, HIGH_PERF	99.0	1173.7	38.8	30.0

8.2 WakeNet

8.2.1 Resource Consumption

Model Type	RAM	PSRAM	Average Running Time per Frame	Frame Length
Quantised WakeNet8 @ 2 channel	50 KB	1640 KB	10.0 ms	32 ms
Quantised WakeNet9 @ 2 channel	16 KB	324 KB	3.0 ms	32 ms
Quantised WakeNet9 @ 3 channel	20 KB	347 KB	4.3 ms	32 ms

8.2.2 Performance Test

Distance	Quiet	Stationary Noise (SNR = 4 dB)	Speech Noise (SNR = 4 dB)	AEC Interruption (-10 dB)
1 m	98%	96%	94%	96%
3 m	98%	96%	94%	94%

False triggering rate: once in 12 hours

Note: The above test results are based on the ESP32-S3-Korvo V4.0 development board and the WakeNet9 (Alexa) model.

8.3 MultiNet

8.3.1 Resource Consumption

Model Type	Internal RAM	PSRAM	Average Running Time per Frame	Frame Length
MultiNet 4	16.8KB	1866 KB	18 ms	32 ms
MultiNet 4 Q8	10.5 KB	1009 KB	11 ms	32 ms
MultiNet 5 Q8	16 KB	2310 KB	12 ms	32 ms
MultiNet 6	32 KB	4100 KB	12 ms	32 ms
MultiNet 7	18 KB	2920 KB	11 ms	32 ms

8.3.2 Word Error Rate Performance Test

Model Type	librispeech test-clean	librispeech test-other
MultiNet5-en	16.5%	41.4%
MultiNet6-en	9.0%	21.3%
MultiNet7-en	8.5%	21.3%

8.3.3 Speech Commands Performance Test

Model Type	Distance	Quiet	Stationary Noise (SNR=5~10dB dB)	Speech Noise (SNR=5~10dB dB)
MultiNet 5_en	3 m	95.4%	85.9%	82.7%
MultiNet 6_en	3 m	96.8%	87.9%	85.5%
MultiNet 7_en	3 m	97.2%	92.3%	90.6%

8.4 TTS

8.4.1 Resource Consumption

Flash image size: 2.2 MB

RAM runtime: 20 KB

8.4.2 Performance Test

CPU loading test (ESP32 @240 MHz):

Speech Rate	0	1	2	3	4	5
Times faster than real time	4.5	3.2	2.9	2.5	2.2	1.8

Chapter 9

Test Method and Test Report

To ensure the DUT performance, some tests can be performed to verify the following parameters:

- Wake-up rate
- Speech recognition rate
- False wake-up rate
- Response Accuracy Rate Under Playback
- Response time

9.1 Test Room Requirement

These tests must be performed in a proper test room. The requirements for this test room include:

- **Size**
  - Area: no smaller than 4 m * 3.2 m
  - Height: no lower than 2.3 m
- **Setup**
  - The floor should be equipped with carpet, the ceiling should be equipped with common acoustic damping materials, and the wall should have 1 to 2 walls with curtains to prevent strong reflection.
  - Room reverberation time (RT60) within the range of [125, 8k] shall be within 0.2 - 0.7 seconds.
  - Do not use anechoic chamber.
- **Background noise**: must < 35 dBA, best < 30 dBA
- **Temperature and humidity**: 20±10°C, 50%±20%
- **Placement of DUT, external noise and voice**:
  - Place the DUT, external noise and voice according the actual use scenario of your DUT.

Note: The RT60, background noise, and the placement of DUT, external noise and voice should be kept the same in all tests.

9.2 Test Case Design

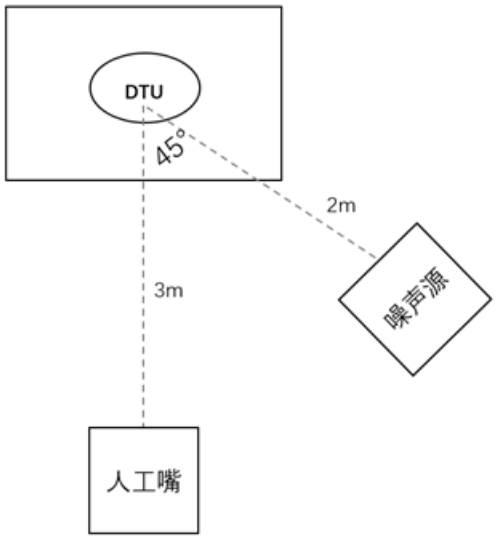
When designing test cases, it’ s suggested to factor in **some or all of the following parameters** based on the actual use scenarios of the product. For example,

- **Different types of noises**
  - White noise
  - Human noise
  - Music

  - News
  - ……
  - Test cases with multiple noise sources can also be added when necessary
- **Different noise levels**
  - < 35 dBA
  - 45 dBA
  - 55 dBA
  - 65 dBA
- **Different voice levels**
  - 54 dBA
  - 59 dBA
  - 64 dBA
- **Different SNR**
  - 9 dBA
  - 4 dBA
  - -1 dBA

9.3 Espressif Test and Result

In all the tests described in this section, the placement of DUT, external noise and voice can be seen in the diagrams below.



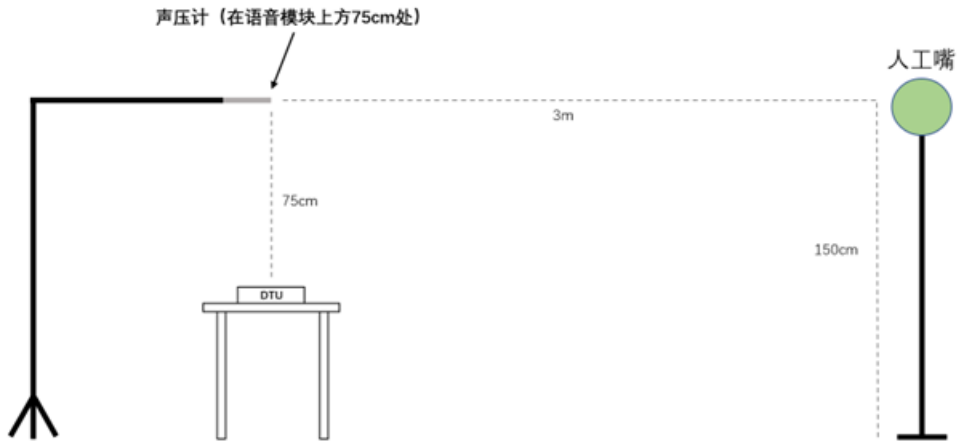
As seen in the diagrams above, place

- The DUT 0.75 meters above the ground.
- The voice 3 meters away from the DUT and 1.5 meters above the ground.
- The external noise 45°C apart from the voice, 2 meters away from the DUT and 1.2 meters above the ground.
- The sound pressure meter right above the DUT by 0.75 meters.

9.3.1 Wake-up Rate Test

Wake-up rate: the probability of the DUT correctly wakes up to a wake word.

Espressif' s Wake-up Rate Test and Result



Test Case	Noise Type	Noise Decibel	Voice Decibel	SNR	Wake-up Rate
1	/	/	59 dBA	/	99%
2	White noise	55 dBA	59 dBA	>= 4 dBA	99%
3	Human noise	55 dBA	59 dBA	>= 4 dBA	99%

9.3.2 Speech Recognition Rate Test

Speech recognition rate: the probability of the DUT correctly recognizes the established command words when the DUT is in the speech recognition state.

Espressif's Speech Recognition Rate Test and Result

Test Case	Noise Type	Noise Decibel	Voice Decibel	SNR	Speech Recognition Rate
1	/	/	59 dBA	/	91.5%
2	White noise	55 dBA	59 dBA	>= 4 dBA	78.25%
3	Human noise	55 dBA	59 dBA	>= 4 dBA	82.77%

9.3.3 False Wake-up Rate Test

False wake-up rate: the probability of the DUT incorrectly wakes up to a random word (that is not a wake word).

Espressif's False Wake-up Rate Test and Result

Test Case	Noise Type	Noise Decibel	Test Duration	Number of False Wake-up
1	Music	55 dBA	12 hours	1 time
2	News	55 dBA	12 hours	1 time

9.3.4 Response Accuracy Rate Under Playback

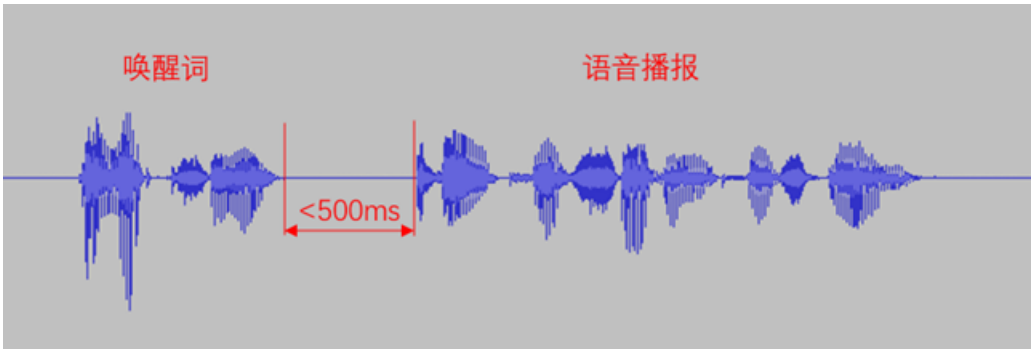
Interrupting wake-up rate: the probability of the DUT correctly responds to a wake word or a command word while playing sounds, such as music or TTS. This test is required for products with AEC feature.

Espressif's Interrupting Wake-up Rate Test and Result

Test Case	Noise Type	Noise / Voice Decibel	SNR	Wake-up Rate	Speech Recognition Rate
1	Music	69 dBA / 59 dBA	>= 10 dBA	100%	96%
2	TTS	69 dBA / 59 dBA	>= 10 dBA	100%	96%

9.3.5 Response Time Test

Response time: the time required for the DUT to respond to a command word. It's measured as the time duration after a command word and before the DUT starts playing sound (see the diagram below).



Espressif's Response Time Test and Result

Test Case	Noise / Voice Decibel	SNR	Response Time
1	NA / 59 dBA	/	< 500 ms

Chapter 10

Glossary

10.1 General Terms

ESP-SR reuses most of its terms in Espressif Advanced Development Framework. See details in ADF English-Chinese Glossary .

10.2 Unique Terms

ESP-SR’s unique terms are listed below.

Voice-User Interface (VUI) 语音用户界面