智慧工廠 - 聲音降噪

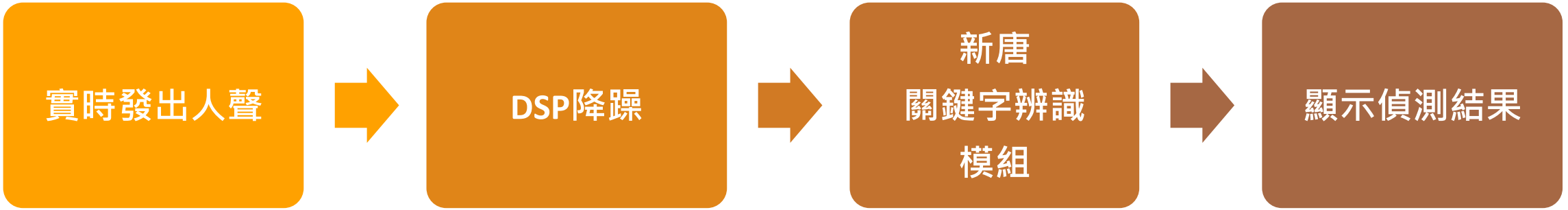
 | Photo by Jeffery Cheng

Outline

- 貳、智慧工廠3 - 聲音降噪
  - 一、案例介紹
  - 二、開發流程概述
  - 三、頻譜減法的理論基礎
  - 四、Python 降噪模擬
  - 五、C 語言在 M55M1 開發板上的實作
  - 六、程式架構與最佳化策略
  - 七、結論與未來展望
  - DEMO 影片

一、案例介紹 - 聲音降噪

- **專案摘要**
- 本專案首先使用 Python 模擬頻譜減法的實作利用頻譜減法(spectrum subtraction)演算法與 Helium 技術實現聲音降躁並成功應用於關鍵字辨識



一、案例介紹 - 聲音降噪

- **專案背景**
- 在數位化與智慧化快速發展的時代，聲音處理技術已成為物聯網、智慧工廠以及嵌入式應用場景中的一項關鍵技術。
- 隨著智慧設備的廣泛應用，用戶對語音交互的需求與日俱增。然而，在工業環境或家庭場景中，背景噪聲干擾往往使語音訊號的準確識別變得困難，這對智慧設備的使用體驗提出了更高的要求。

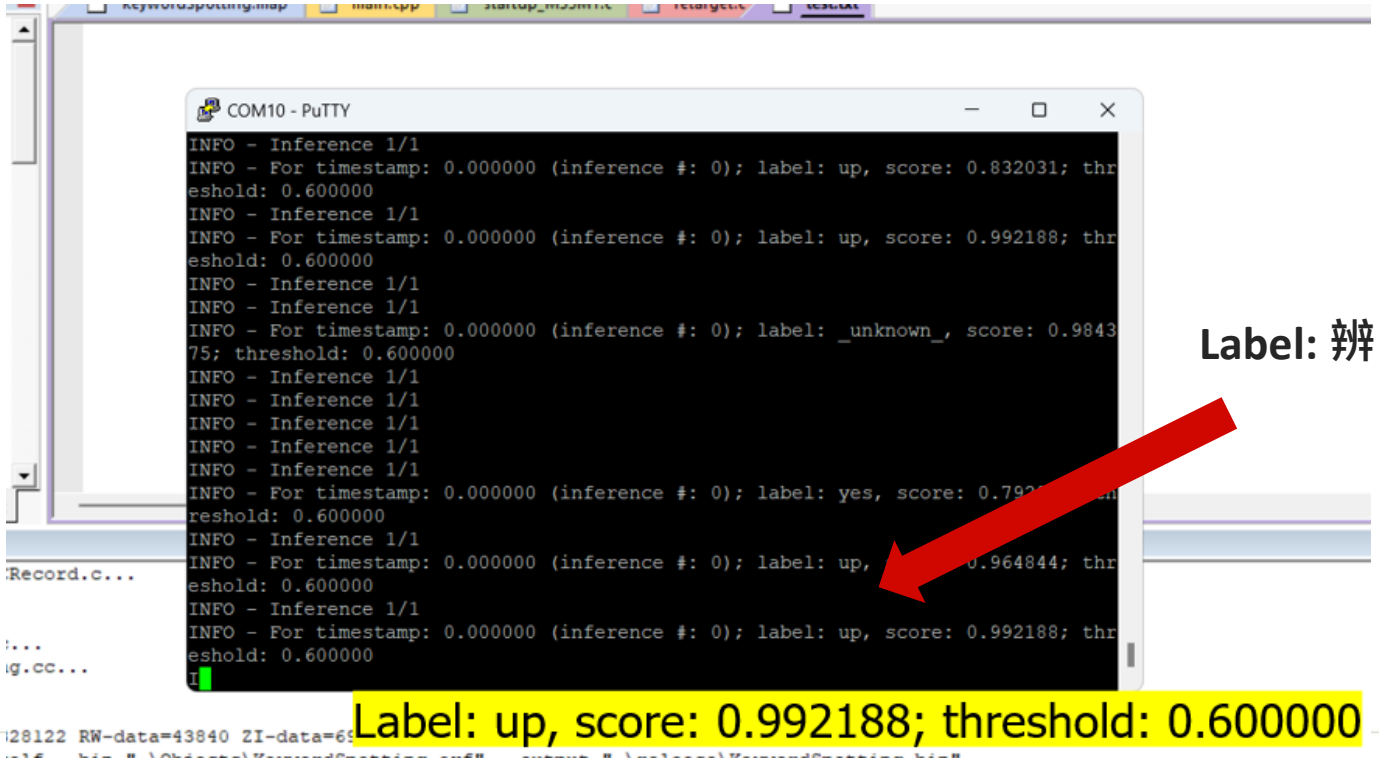
一、案例介紹 - 聲音降噪

- **預期目標**
- 本專案基於新唐提供的關鍵字辨識模組，該模組能夠辨識十種常用關鍵字，包括："yes"、"no"、"up"、"down"、"left"、"right"、"on"、"off"、"stop"、"go"。
- 引入透過 ARM Helium 技術加速運算的頻譜減法降噪功能，提升系統性能。
- 增強終端機的交互功能，提供更豐富的操作選項以便於使用者控制與調整。

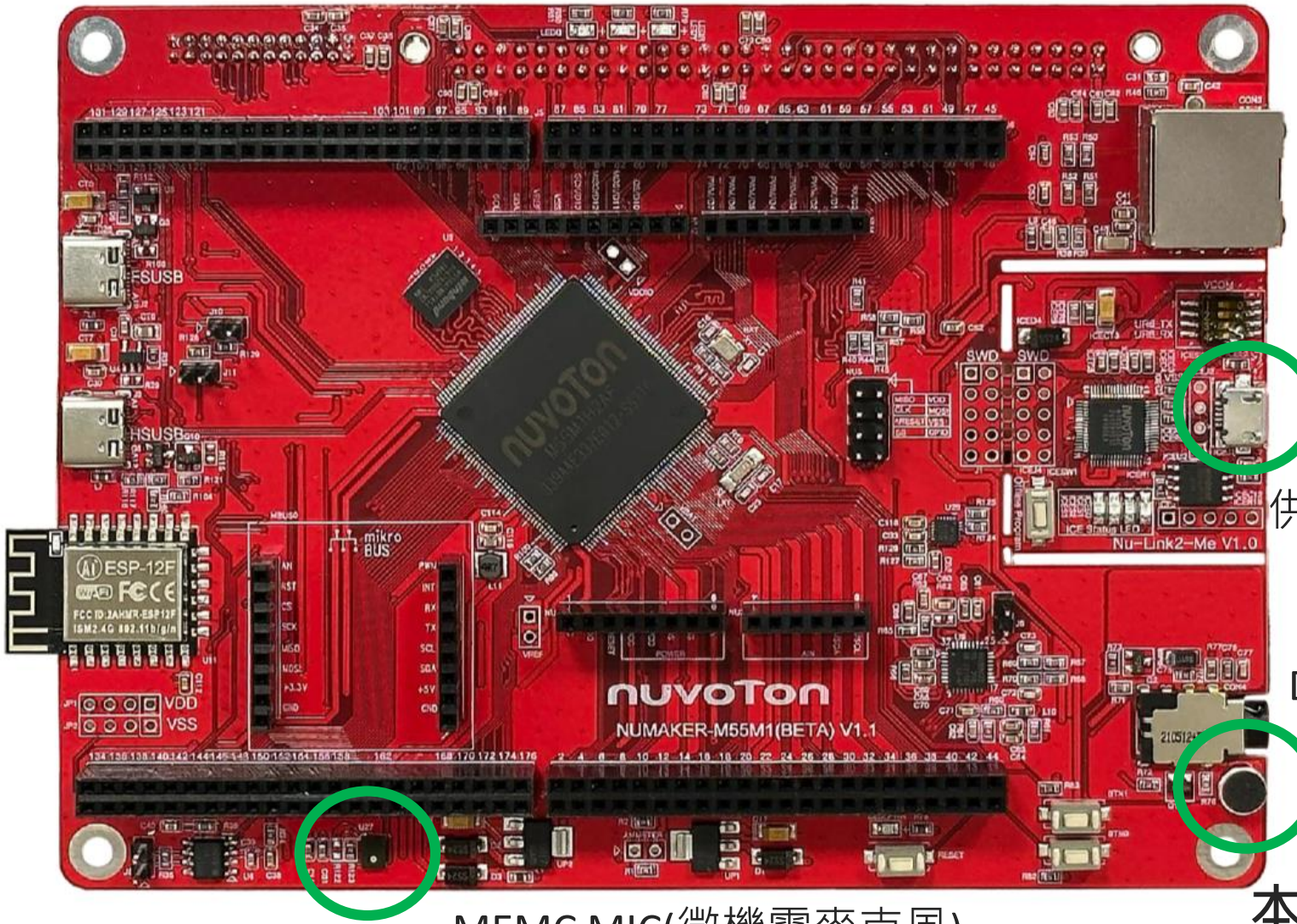
M55M1 開發板

一、案例介紹 - 聲音降噪

- 實際呈現結果
  - 原始的關鍵字辨識模組在噪音干擾環境下測試的準確率為 69%（69/100）
  - 透過加入頻譜減法的降噪功能後，辨識準確率顯著提升至 80%（80/100）



一、案例介紹 – 專案相關I/O模組



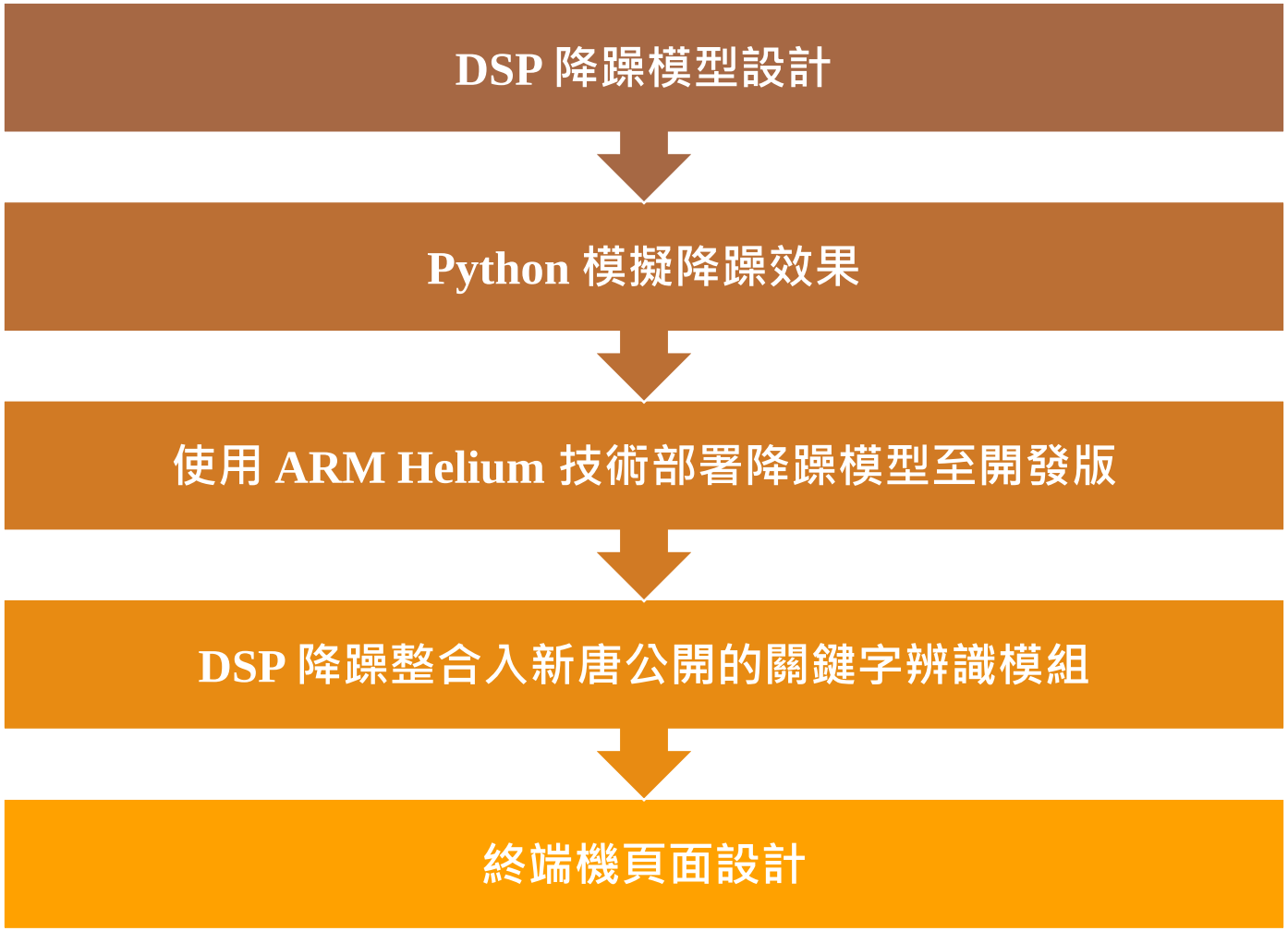
供電模組與UART傳輸設備

DMIC(數位麥克風)

本題目使用此麥克風

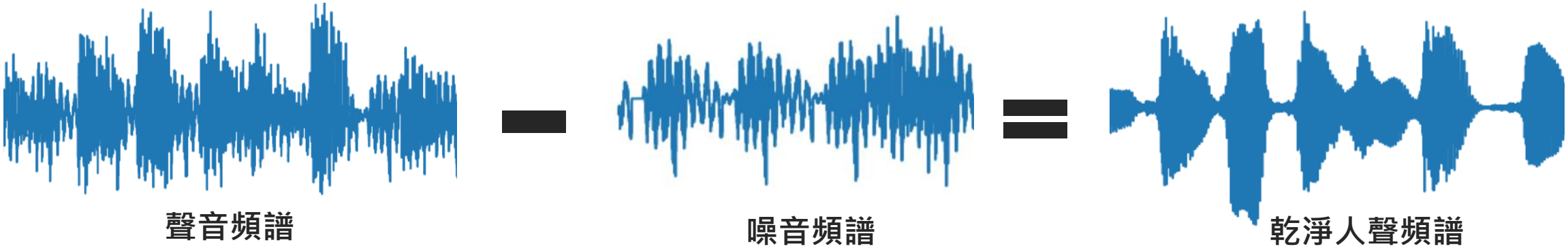
MEMC MIC(微機電麥克風)

二、開發流程概述



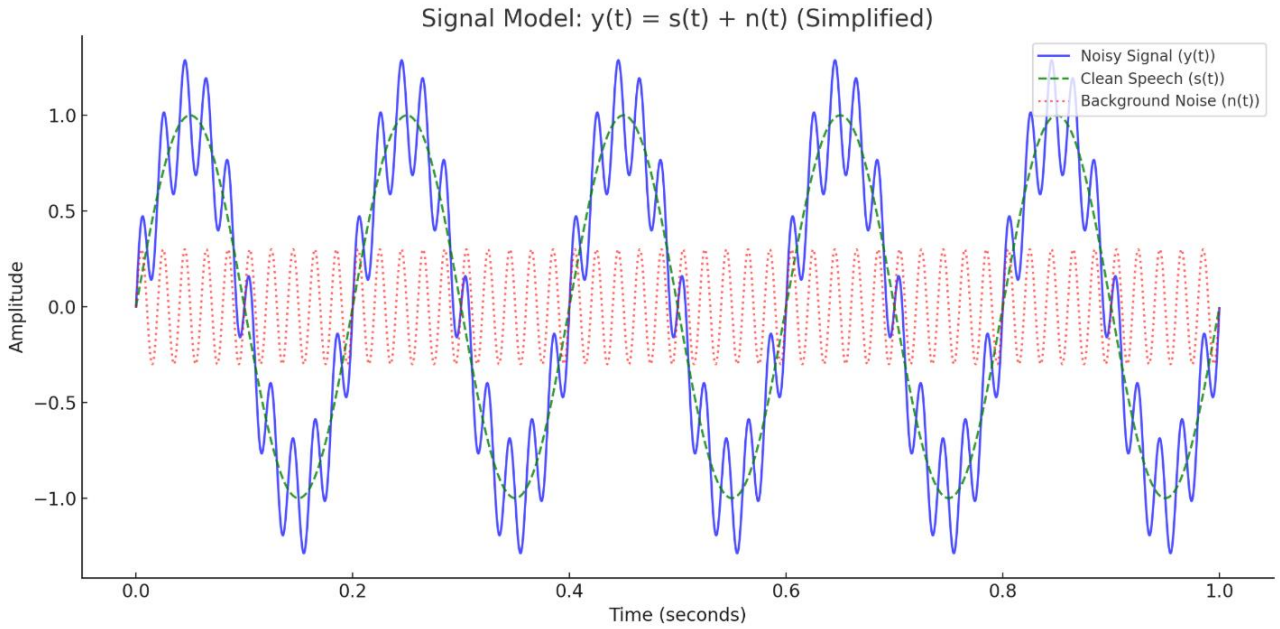
三、頻譜減法的理論基礎

- 核心概念
  - 分析噪音與語音的頻譜
  - 從語音頻譜中減去背景噪音的頻譜
  - 重建降躁後的聲音波型



三、頻譜減法的理論基礎

- **信號模型**
  - 常見的含噪語音信號通常由乾淨語音與背景噪聲疊加而成
  - y(t)=s(t)+n(t)
    - y(t)：含噪語音信號
    - s(t)：語音信號
    - n(t)：背景噪聲



y(t)=s(t)+n(t) 示意圖

三、頻譜減法的理論基礎

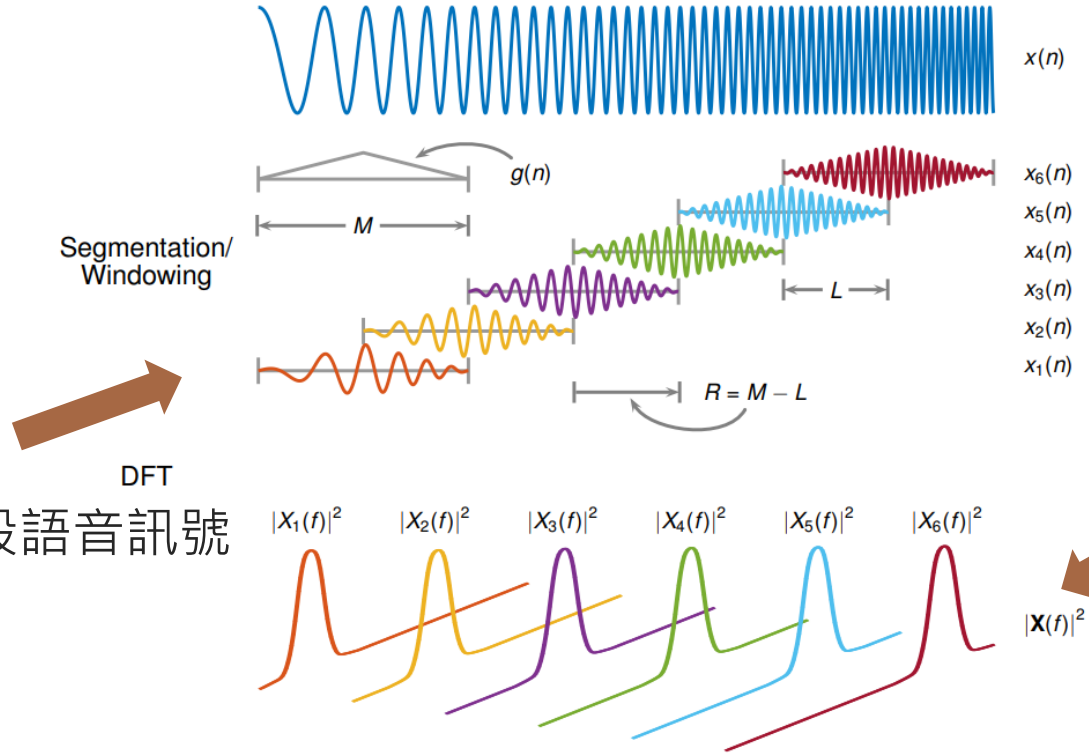
- **短時傅立葉變換（STFT）**
  - 語音信號是時變的，直接使用整體傅立葉變換分析頻譜可能導致結果失準。
  - 將信號切分為小片段，逐段計算頻譜，結合時間與頻率特性進行分析，可有效解決此問題。
  - 簡單來說，就是將完整聲音分割為多段 y(t)，逐段計算其頻譜 Y(f)。

$$STFT\{y(t)\} = Y(f,t) = \int_{-\infty}^{\infty} y(\tau) w(\tau - t) e^{-j2\pi f \tau} d\tau$$

STFT數學公式

三、頻譜減法的理論基礎

- 短時傅立葉變換（STFT）



將整段聲音切割成多段語音訊號

進行傅立葉轉換得到聲音頻譜

https://www.mathworks.com/help/signal/ref/stft.html

三、頻譜減法的理論基礎

- **頻譜減法公式**
  - 將噪聲看作是一層汙漬，頻譜減法的作用就像是抹布，將汙漬去除，露出清晰的語音。
  - $|S(f)| = |Y(f)| - \alpha|N(f)|$

    α：噪聲縮放因子，控制減噪強度

    $|Y(f)|$：含噪信號的頻譜（Magnitude Spectrum of Noisy Signal）

    $|S(f)|$：純淨語音的頻譜（Magnitude Spectrum of Speech）

    $|N(f)|$：背景噪聲的頻譜（Magnitude Spectrum of Noise）

三、頻譜減法的理論基礎

- **語音重建**
  - 用相位填補去噪後的頻譜，將信號還原回時域。
  - 去噪語音的幅度頻譜 S(f)

    $|S(f)|=\max(|Y(f)|-\alpha|N(f)|,0)$
  - 經過相位信息與 ISTFT 重建的語音信號 s(t)

    $s(t) = \mathrm{ISTFT}(|S(f)|e^{j\phi Y})$

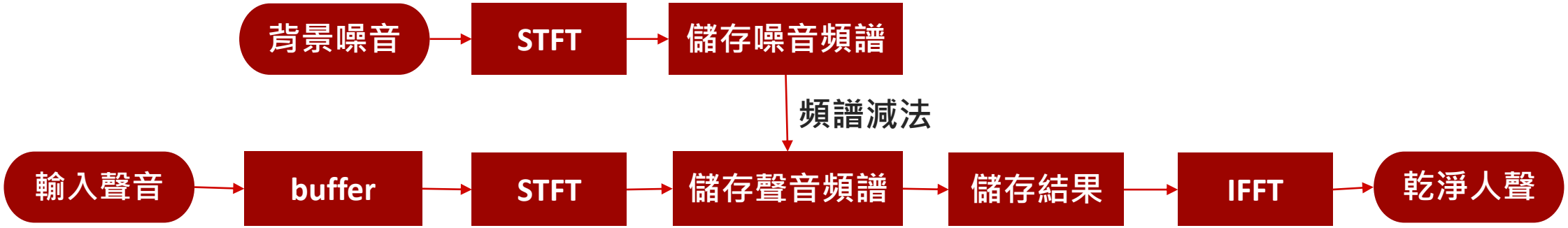
    ϕ^{Y}：含噪信號的相位（Phase of Noisy Signal）

四、Python 降噪模擬

- **模擬目的**：使用 Python 簡單實現頻譜減法，直觀展示降噪效果。
- Python **的優勢**：提供方便的音訊處理工具
- **語音處理相關處理庫**
  - Librosa：提供便捷的音訊讀取、STFT、頻譜處理和重建功能。
  - Soundfile：支持高效音訊文件的讀寫，適合處理大型音訊數據。
  - Scipy：供傅立葉變換、濾波器設計等數學工具。
  - Matplotlib：繪製波形圖、頻譜圖等視覺化效果，清晰展示音訊特性。

四、Python 降噪模擬

- **實作步驟**
  1. 收集背景噪音作為基準
  2. 將語音進行 STFT（短時傅立葉轉換），轉換到頻域。
  3. 使用噪音頻譜減去語音頻譜。
  4. 應用 IFFT（逆傅立葉變換）重建降躁後的聲音。



四、Python 降噪模擬

- 測試音訊生成
  - **Mozilla Common Voice (MCV)**

    包含多達2454小時的錄音資料，分佈在簡短的**MP3**檔案中。資料集的一個特點是說話者的多樣性，包括不同年齡和口音的男性和女性。
  - **UrbanSound8K**

    包含8732個標記的聲音樣本，涵蓋了城市中常見的**10**種聲音，如喇叭聲、警笛聲等。

    UrbanSound8K作為噪音訊號添加到**MCV**中的語音訊號中，就可以獲得穩定且多樣化的含噪聲音訊號

四、Python 降噪模擬

- 讀取音訊與噪聲樣本提取

```
# set input and output file paths
input_file = 'mix_noisy.wav'
output_file = 'denoise_spectral.wav'

# load audio
y, sr = librosa.load(input_file, sr=None)

# assume the first 0.5 seconds of the audio is noise
noise_duration = 0.5
noise_sample = y[:int(noise_duration * sr)]
```

使用librosa.load讀取含噪聲音

設定聲音前0.5秒為噪音樣本

四、Python 降噪模擬

- 計算噪聲頻譜與參數調整

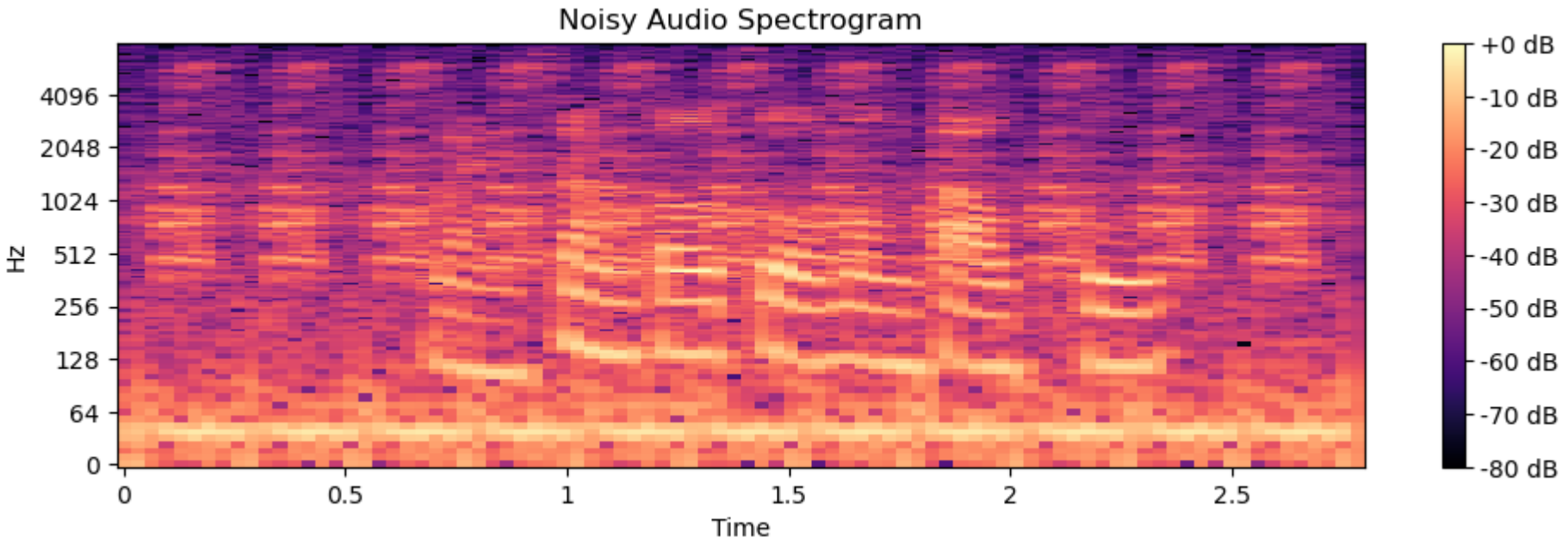
```
17  # compute the noise spectrum
18  noise_stft = librosa.stft(noise_sample)
19  # average the magnitude spectrum of the noise
20  noise_factor = 2.5  # factor to reduce noise
21  mean_noise_spectrum = np.mean(np.abs(noise_stft), axis=1) * noise_factor
22  # STFT
23  y_stft = librosa.stft(y)
24  magnitude, phase = np.abs(y_stft), np.angle(y_stft)
```

α：縮放因子(預設為 2.5)
頻譜減法公式

使用librosa.stft計算聲音頻譜

四、Python 降噪模擬

- 計算噪聲頻譜與參數調整



視覺化的噪音頻譜

四、 Python 降噪模擬

- 音訊重建

```
# reconstruct the denoised audio
y_stft_denoised = magnitude_denoised * np.exp(1j * phase)
# inverse STFT
y_denoised = librosa.istft(y_stft_denoised)
# save the denoised audio
sf.write(output_file, y_denoised, sr)
```

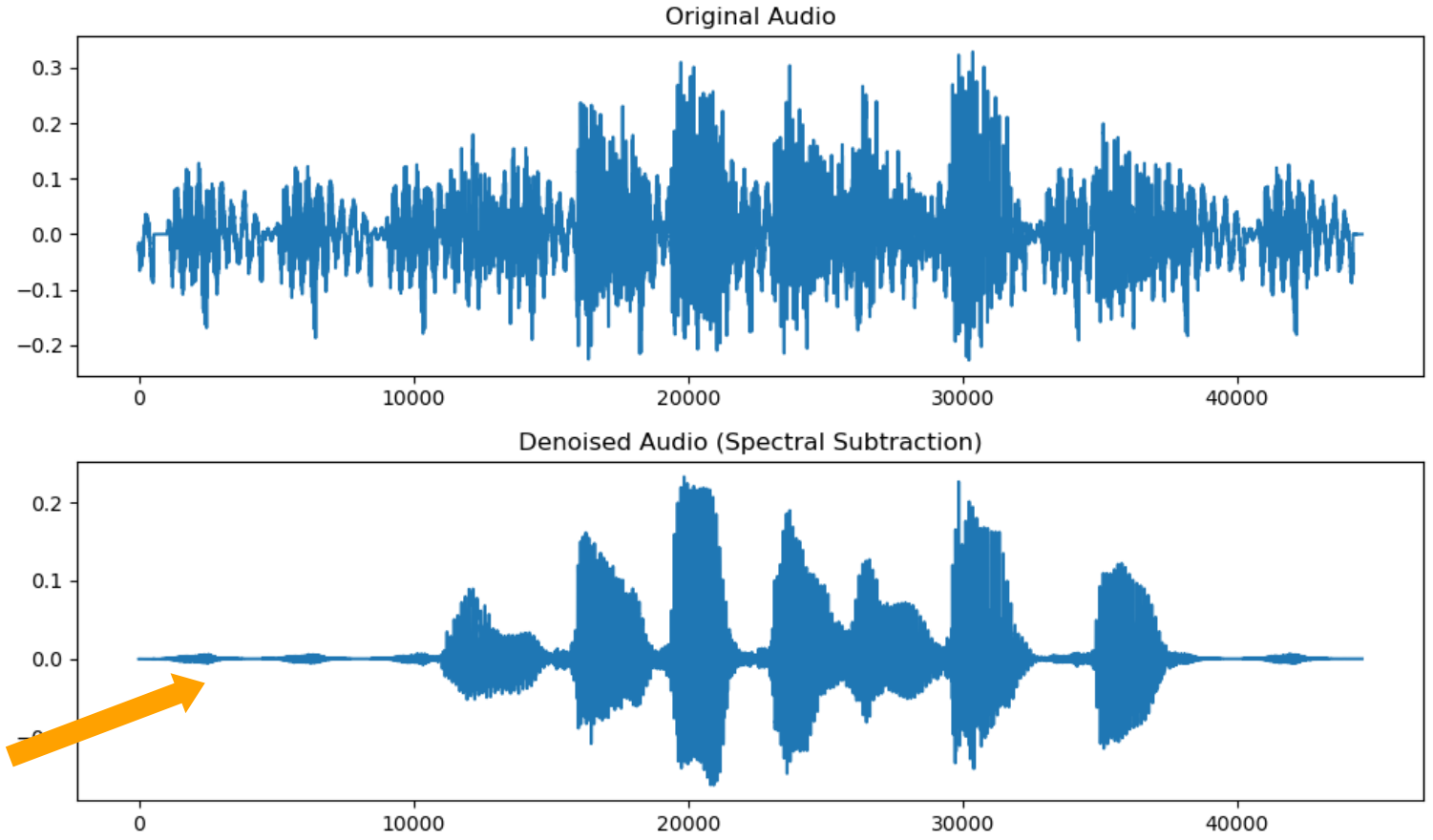
重新建構聲音頻譜的相位

使用librosa.istft計算降噪後的聲音訊號

使用sf.write將降噪後的聲音訊號儲存

四、 Python 降噪模擬

- 視覺化降噪前後聲音波形



噪音明顯消除

四、 Python 降噪模擬

- **STOI (Short-Time Objective Intelligibility)**
  - STOI 是一種常用的語音品質評估指標，用於測量含噪語音或經過處理的語音相對於乾淨語音的可懂度。
  - 分數範圍：0（完全不可懂）到 1（完全可懂）。
  - 特別適合評估降噪和語音增強技術的效果。

在python中可以使用pystoi中的stoi函式快速計算出STOI的分數

```
28  # compute the STOI score
29  stoi_score = stoi(clean, denoised, sr_clean, extended=False)
```

➡ 通過該方法進行降噪後，獲得 STOI 分數：0.9
表明處理後的語音接近乾淨語音，顯示出降噪效果非常顯著。

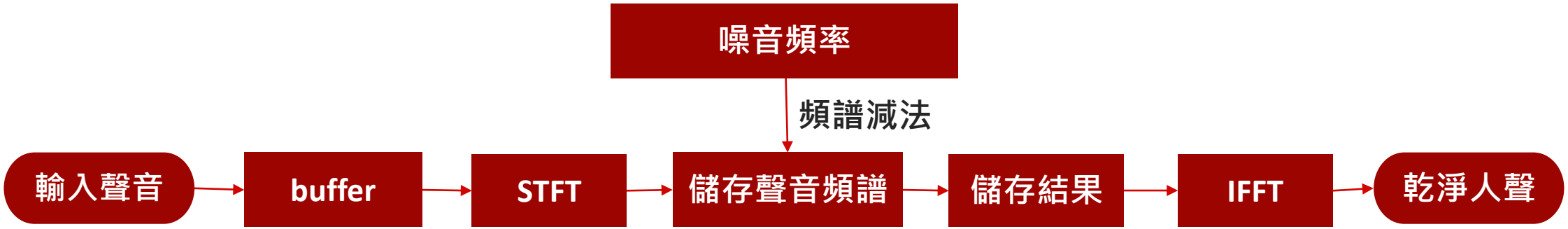
五、C語言在M55M1開發板上的實作

五、C語言在M55M1開發板上的實作

- **ARM Helium 技術簡介**
- Helium 是 ARM 的 MVE（M-Profile Vector Extension）功能，專為 DSP（數位訊號處理）和 ML（機器學習）任務優化。
- 提供高效能、低功耗的計算能力，專注於嵌入式系統應用。
- 現階段只有Cortex-M55等高階CPU支援此功能。

五、C語言在M55M1開發板上的實作

- 降噪模型設計



五、C語言在M55M1開發板上的實作

- 計算聲音頻譜

```
51    // PI_VALUE
52    const float PI_VALUE = 3.14159265358979323846f;
53    void ApplySTFTDenoising(const std::vector<int16_t>& inputAudio,
54                            std::vector<int16_t>& outputAudio) {
55        size_t frameSize = 1024;  // frame size
56        size_t hopSize = frameSize / 2;  // overlap size
57        std::vector<float> frame(frameSize);  // store time-domain frame data
58        std::vector<float> fftOutput(frameSize);  // store FFT result
59
60        // initialize FFT
61        arm_rfft_fast_instance_f32 fftInstance;
62        arm_rfft_fast_init_f32(&fftInstance, frameSize);  // initialize FFT
63
64        // make sure the output audio size is the same as the input
65        outputAudio.resize(inputAudio.size(), 0);
```

初始化傅立葉轉換參數

五、C語言在M55M1開發板上的實作

- 計算聲音頻譜

```
67      for (size_t i = 0; i + frameSize <= inputAudio.size(); i += hopSize) {
68          // 1. convert the audio data of the current frame to float
69          for (size_t j = 0; j < frameSize; ++j) {
70              frame[j] = static_cast<float>(inputAudio[i + j]);
71          }
72          // 2. apply Hamming window
73          for (size_t j = 0; j < frameSize; ++j) {
74              frame[j] *= 0.54f - 0.46f * cos(2 * PI_VALUE * j / (frameSize - 1));
75          }
76          // 3. perform FFT
77          arm_rfft_fast_f32(&fftInstance, frame.data(), fftOutput.data(), 0);
78          // 4. subtract the spectrum of the current frame from the background spectrum
```

設定切隔聲音的窗隔大小並儲存

進行傅立葉轉換

“0”表示執行傅立葉轉換

五、C語言在M55M1開發板上的實作

- 進行頻譜減法並還原成時域的聲音

```
79          float noiseThreshold = 0.1f;  // noise threshold
80          for (size_t j = 0; j < fftOutput.size(); ++j) {
81              if (fftOutput[j] < noiseThreshold) {
82                  fftOutput[j] = 0;  // set the value to 0 if it is less than the threshold
83              }
84          }
85          // 5. perform inverse FFT (iFFT) to convert the spectrum data back to time domain
86          arm_rfft_fast_f32(&fftInstance, fftOutput.data(), frame.data(), 1);
87          // 6. copy the reconstructed frame data back to the output, perform overlap-add
88          for (size_t j = 0; j < frameSize; ++j) {
89              outputAudio[i + j] += static_cast<int16_t>(frame[j]);
90          }
91      }
92  }
```

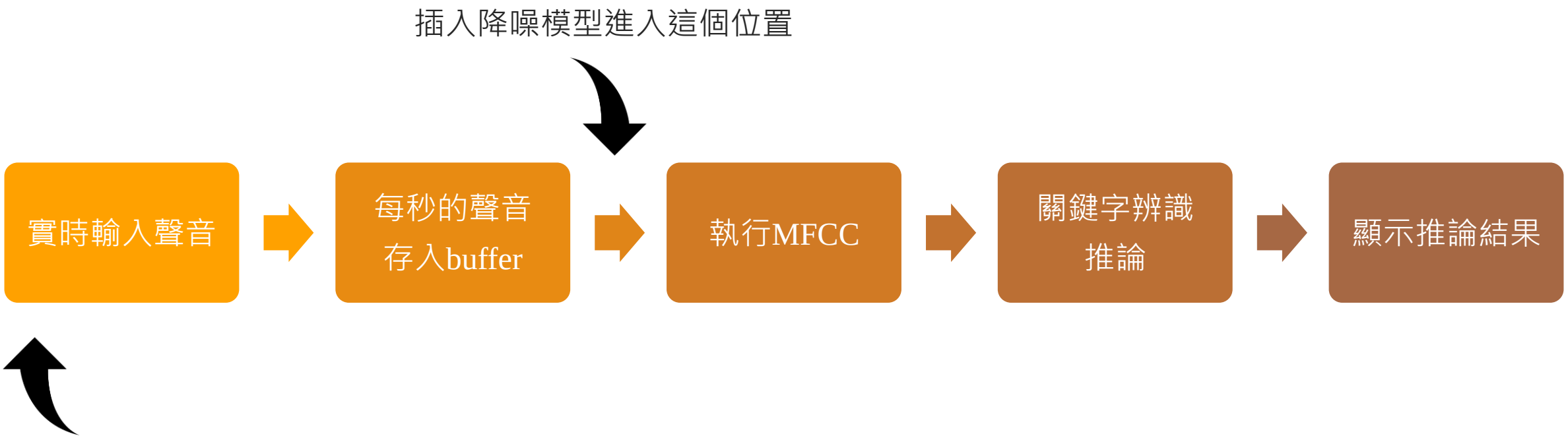
刪除噪音頻率的部分

執行逆傅立葉轉換並合併每段聲音成完整乾淨的人聲

1表示執行逆傅立葉轉換

六、 關鍵字辨識模組整合

六、 關鍵字辨識模組整合



六、 關鍵字辨識模組整合

- 輸入’y’開始包含降噪的關鍵字辨識

```
// Start keyword spotting
printf("Type 'y' to start keyword spotting with DSP Noisy Suppression\n");
```

終端機顯示Type ‘y’ to start keyword spotting with Noisy Suppression

```
while (1) {
    char c = getchar();  // get UART input character
    if (c == '\r' || c == '\n') {
        uartInput[inputIndex] = '\0';
        if (strcmp(uartInput, "y") == 0) {
            printf("Starting keyword spotting...\n");
            break;
        } else {
            printf("Invalid input. Please type 'y'.\n");
            inputIndex = 0;
            memset(uartInput, 0, sizeof(uartInput));
        }
    } else {
        if (inputIndex < sizeof(uartInput) - 1) {
            uartInput[inputIndex++] = c;
        }
    }
}
```

等待收到’y’跳離迴圈

六、 關鍵字辨識模組整合

- 聲音降噪

將由DMIC收入的聲音分段收錄至buffer

```
411         // todo: DSP denoising
412         // source audio data
413         std::vector<int16_t> inputAudio(audioSlidingBuf, audioSlidingBuf + audioSlidingSamples);
414         // store the denoised audio data
415         std::vector<int16_t> denoisedAudio;
416         //* Normal case
417         ApplySpectralSubtraction(inputAudio, denoisedAudio, backgroundSpectrum);
```

進行DSP聲音降噪

六、 關鍵字辨識模組整合

- 進入MFCC處理，開始關鍵字辨識

```
584                 info("Inference %zu/%zu\n", audioDataSlider.Index() + 1,
585                      audioDataSlider.TotalStrides() + 1);
586
587     #if defined(__PROFILE__)
588                 u64StartCycle = pmu_get_systick_Count();
589     #endif
590
591                 /* Run the pre-processing, inference and post-processing. */
592                 if (!preProcess.DoPreProcess(inferenceWindow, arm::app::audio::KwsMFCC::ms_defaultSamplingFreq))
593                 {
594                     printf_err("Pre-processing failed.");
595                     break;
596                 }
```

經過降噪處理後就能進入新堂設計的關鍵字辨識功能

七、結論與未來展望

- **結論**
  - **核心技術整合**

    系統結合背景噪音錄製、實時音訊處理、關鍵字偵測與降噪技術。發揮 Helium 指令集優化能力，實現高效的 STFT 和頻譜減法處理。結合 MFCC 特徵提取與機器學習模型，成功達到高準確度的關鍵字識別。
  - **系統設計與特性**

    模組化結構設計，確保程式的可讀性與易於維護。強調系統邏輯清晰的流程，適應多樣化環境噪音，提高應用穩定性與實用性。
  - **設計成果**

    系統適合嵌入式語音處理應用，具備高效能、高準確度的關鍵字偵測功能。 提供了可擴展性架構，為後續研究奠定了穩固的基礎。

七、結論與未來展望

- **未來展望**
  - **功能模組擴展**：

    新增噪音分類功能，提升降噪模組的適應性與泛用性。
  - **進階技術結合**：

    引入深度學習降噪技術，提高系統在複雜噪音環境下的降噪效果。
  - **應用領域拓展**：

    系統可應用於語音助理、智能家居控制、遠場語音識別等更多場景，進一步發揮其嵌入式語音智能應用的潛力。

DEMO 影片

Link

Joy of innovation

NUVOTON

谢谢

謝謝

Děkuji

Bedankt

Thank you

Kiitos

Merci

Danke

Grazie

ありがとう

감사합니다

Dziękujemy

Obrigado

Спасибо

Gracias

Teşekkür ederim

Cảm ơn