Department of Information Engineering and Computer Science

Bachelor's Degree in
Computer, Communication and Electronic Engineering

Final Dissertation

TinyML-Based Voice Recognition on Syntiant NDP101

From Keyword Spotting to Speaker Verification

Supervisor
Kasim Sinan Yildirim

Student
Matteo Gottardelli

Academic year 2024/2025

Contents

Abstract

ROWS TO ELIMINATE OR ADD: 1. INTRODUCTION: Try fit Section 1.3 in the first page

List of reference explanation:

- Edge Impulse:
  1. Edge Impulse Explaining how to manage Syntiant TinyML Deployment: [?]
  2. Keyword Spotting on Syntiant Stop/Go Example by Edge Impulse: [?]
  3. Firmware Syntiant TinyML Github Repository: [?]
  4. Processing Blocks Edge Impulse Code Processing (No Deployment Model Conversion code): [?]
- Tools and Datasets:
  1. Node Js Repository Download Source: [?]
  2. Dataset Edge Impulse Background Noise or General Words from Google Dataset: [?]
  3. Google Speech Dataset: [?]
  4. Netron Model Representation App: [?]
- Documentation:
  1. ESP32 Documentation: [?]
  2. Syntiant Tutorial Edge Impulse: [?]
- Syntiant Specifics:
  1. TinyML 2021 Summit Presentation Syntiant NDP120 -¿ Model Processing: [?]
  2. Data of Syntiant NDP101: [?]
  3. Code Experimental Implementation on NDP101: [?]
  4. The Intelligence of Things enabled by Syntiant's TinyML board analyzing performances: [?]
  5. NDP101 General Usage: [?]
  6. Description Syntiant Audio Block Processing: [?]
  7. Hardware NDP101 Properties: [?]
- Theory:
  1. PDM Microphone Module Explanation: [?]
  2. MFE Block Process Explained: [?]
  3. TinyML Neural Network Training: [?]
  4. Distillation Method from CNN to DNN: [?]
  5. Knowledge Distillation: [?]
  6. Understanding 4-bit Quantization: [?]

- SV Model:
    1. Deep Neural Network Model for Speaker Identification: [?]
    2. D-vector Extractor implementing TinySV: [?]
    3. D-vector Extractor implementing TinySV Code: [?]

1 Introduction

The microcontrollers (MCU) are computing systems, which are integrated in a larger system and in that case they are called embedded system, to perform a specific function that need a software implementation (programmed in C or assembly) and a hardware one (interconnectivity wires and sensor handle) They operate in a closed environment and elaborate environment inputs which may be visual, acustic and with more recent technology even tactile or movement and elaborate it to generate an output which may be an action in a bigger system, an audio response or a trigger depending on the microcontroller specifics. So, these MCUs work in application-specific fields and they are typically used in real time. The programmer can implement a desired application, which allows personalization of the functionalities and the possibility of optimization and because of that with a correct usage a MCU may be a good addition to reduce costs in power and energy terms, the possibility of build a desired application and adding more than one feature at the same time connecting various sensors thanks to their peripherals. Each MCU can be optimized in a specific sub-application, like monitoring, communications and networking (Internet-of-Thing) and digital signal processing.

1.1 TinyML Concept and Limits

During the years, MCU's technology made steps ahead in optimizing the computation velocity, meanwhile minimizing power consumption and a result is TinyML (Tiny Machine Learning). These devices enable machine and deep learning models to operate on a MCU, enabling more complex program introduction, like Anomaly Detection in images or analog data collection, Keyword Spotting, recognizing a specific word in an audio stream, or identifying objects in an image. These complex functionality can be implemented thanks to a Neural Network, which is trained typically on cloud resources in Python programming language and this leads in performing only the inference phase on these tiny devices. This approach does not allow data exploitation directly, limiting incremental training or adapting algorithm through the device life. This is a limit on the Machine Learning side, but on the Tiny one there are some trade-offs. A direct consequence of being really small devices is having limited memory to reduce power consumption, so sometimes adapting a Neural Network which typically may occupy much memory isn't easy and requires precision reduction.

1.2 Goals - TinySV

This thesis studies how to adapt with a TinyML devices base system that performs Speaker Verification, which task consists in recognizing the identity of a user with references samples and comparing them with an input audio stream. For this goal already exists a solution from which was taken reference[?]. The objective originally was creating a Keyword Spotting model (KWS) and a Speaker Verification (SV) one, trying to adapt that algorithm on two Syntiant TinyML NDP101 devices, but because of a NDA problem the access to device documentation was inaccessible, allowing KWS development thanks to Edge Impulse[?], but SV, on the other hand, uses a technique not supported by Edge Impulse and because of the inability of accessing to a model compression tool, we had to opt on a STM32 to perform a model verification. To preserve the initial idea, the model was tested as it would be a Syntiant TinyML NDP101, to show the validation of the technique. The full logic has to be hardware implemented that could have been possible using Syntiant SDK, but because of this impossibility, with this thesis the objective is to show the feasibility of this idea from a software perspective and partially hardware. For real testing the model verification will be single, but the whole system has been tested with a software code working on Computer-side. All the codes used in this thesis is provided on a GitHub repository[1]

[1]Thesis GitHub Repository - https://github.com/Gotta003/Thesis

1.3 Brief Summary

The thesis is divided into chapters. After this introduction, Chapter 2 aims to present theoretical concepts that will be used in this thesis, like Audio Processing, KWS and SV. Chapter 3 introduces the general methodology of the final objective and explains how it works. Chapter 4 explains the work flow of models training and optimization. Chapter 5 presents the software C code implementation for deploying both algorithms. Chapter 6 draws up the results obtained from computer testing.

2 Introduction

The microcontrollers (MCUs) are computing systems which are integrated into a larger system, and in that case, they are called embedded systems. They perform a specific function that requires both a software implementation (programmed in C or assembly) and a hardware one (interconnectivity wires and sensor handling). They operate in a closed environment and process environmental inputs, which may be visual, acoustic, or—thanks to more recent technology—even tactile or movement-based, and elaborate them to generate an output. This output may be an action in a larger system, an audio response, or a trigger, depending on the microcontroller's specifics. Thus, these MCUs work in application-specific fields and are typically used in real-time applications. The programmer can implement a desired application, which allows for the personalization of functionalities and the possibility of optimization. Because of that, with proper usage, an MCU may be a good addition to reduce costs in terms of power and energy, to build a desired application, and to add multiple features simultaneously by connecting various sensors through their peripherals. Each MCU can be optimized for a specific sub-application, such as monitoring, communications and networking (Internet-of-Things), or digital signal processing.

2.1 TinyML Concept and Limits

Over the years, MCU technology has made significant progress in optimizing computational speed while minimizing power consumption, and one result of this is TinyML (Tiny Machine Learning). These devices enable machine and deep learning models to operate on an MCU, allowing for the implementation of more complex programs such as anomaly detection in images or analog data, keyword spotting (recognizing a specific word in an audio stream), or object identification in images. These complex functionalities can be implemented thanks to a neural network, which is typically trained using cloud resources in the Python programming language. This leads to performing only the inference phase on these tiny devices. This approach does not allow direct data exploitation, limiting incremental training or algorithm adaptation over the device's life. This is a limitation on the machine learning side, but there are trade-offs on the TinyML side as well. A direct consequence of being very small devices is having limited memory to reduce power consumption, so adapting a neural network—which typically occupies much memory—is not easy and often requires precision reduction.

2.2 Goals - TinySV

This thesis studies how to adapt a base system with TinyML devices that performs speaker verification, a task that consists of recognizing the identity of a user through reference samples and comparing them with an input audio stream. For this goal, an existing solution was used as a reference[**?**]. The original objective was to create both a keyword spotting (KWS) model and a speaker verification (SV) model, and to try to adapt that algorithm to two Syntiant TinyML NDP101 devices. However, due to an NDA problem, access to the device documentation was unavailable. This allowed for KWS development thanks to Edge Impulse[**?**], but SV, on the other hand, uses a technique not supported by Edge Impulse. Due to the inability to access a model compression tool, we had to opt for an STM32 to perform model verification. To preserve the initial idea, the model was tested as if it were running on a Syntiant TinyML NDP101 device to demonstrate the validation of the technique. The full logic should be hardware-implemented, which could have been possible using the Syntiant SDK. However, due to this impossibility, this thesis aims to show the feasibility of the idea from a software perspective and partially from a hardware one. For real testing, model verification will be done individually, but the whole system has been tested with software code running on the computer side. All the code's files created for this thesis are provided in a GitHub repository[1]

2.3 Brief Summary

The thesis is divided into chapters. After this introduction, Chapter 2 presents theoretical concepts used in this thesis, such as audio processing, KWS, and SV. Chapter 3 introduces the general methodology of the final objective and explains how it works. Chapter 4 explains the workflow of model training and optimization. Chapter 5 presents the software C code implementation for deploying both algorithms. Chapter 6 presents the results obtained from computer testing.

[1]Thesis GitHub Repository - https://github.com/Gotta003/Thesis

3 Background Notions

3.1 NDP101 Architecture Overview

This is a MCU developed by Syntiant[?], which is a company specialized in Edge AI device development. NDP stands of Neural Decision Processor and is an architecture specialized in deep learning algorithm with audio processing application, like keyword speech interface, sensor applications and speaker identification. The device is composed by two parts, the TinyBoard, which contains the CPU that handles the peripherals, being the hardware part, and a Syntiant NDP101 Core, in which happens all the audio processing and where the Neural Network is stored in. It is important to notice that the DNN (Dense Neural Network) Architecture is fixed, dedicating to 256KB with int32 bias length and int4 weights, which can be at most 589.000, with a total memory. The neural network for this device to be fast enough in computation avoid the use of CNN (Convolutional Neural Network), but can only support 4 Fully Connected Layers, 3 intermediate with 256 neurons and one for output with at most 64 output classes, with classification. To perform the internal software computation there is an internal SRAM inside the chip which is a Cortex-M0 112KB size. This piece of memory contains the binary user's code alongside the global and local variables.

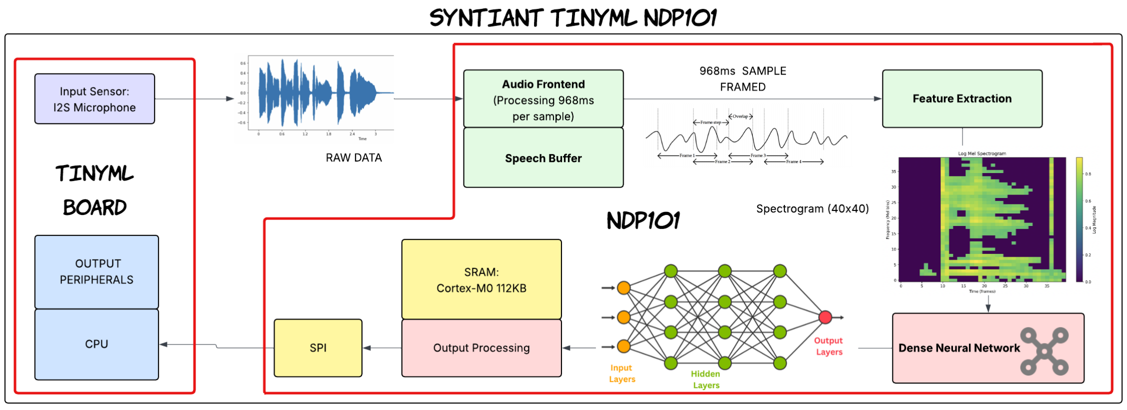

Figure 3.1: Syntiant NDP101 High Level Workflow

A typical behavior of the device is the following. Starting with a I2S microphone input sensor, which if active will be always on, performing a polling behavior. The input is processed by the Audio Front End, which will perform samples of 968ms, in the meanwhile the system feeds a 96KB speech buffer. These samples are gave in input to the feature extraction that acts like a MFE Block Processing[?], using log-mel spectrogram. This spectrogram will always have 1600 features which will correspond to a 40x40 Spectrogram image. This image is processed by the Dense Neural Network. As default Syntiant NDP101 performs classification, but how the output is used is up to the programmer that can write a code to manage the output of the neural network using Arduino IDE. The peripherals of the device, other than the microphone are 9 pins, 4 for power supplying and 5 for GPIO (General Purpose I/O). Other than the Serial Flash Memory (SRAM), there is a micro-SD card slot used for memory extension. In this thesis, we will not use it but for big data storage and for the IMU functionality, supported by the device, that is required and is estimated that with a 32GB card it will save more than 3 days of uncompressed audio data, with a frequency of 16kHz and with IMU more than 300 days of 6-axis sensor data with a frequency of 100Hz. As following there is a image of the principle peripherals connections:

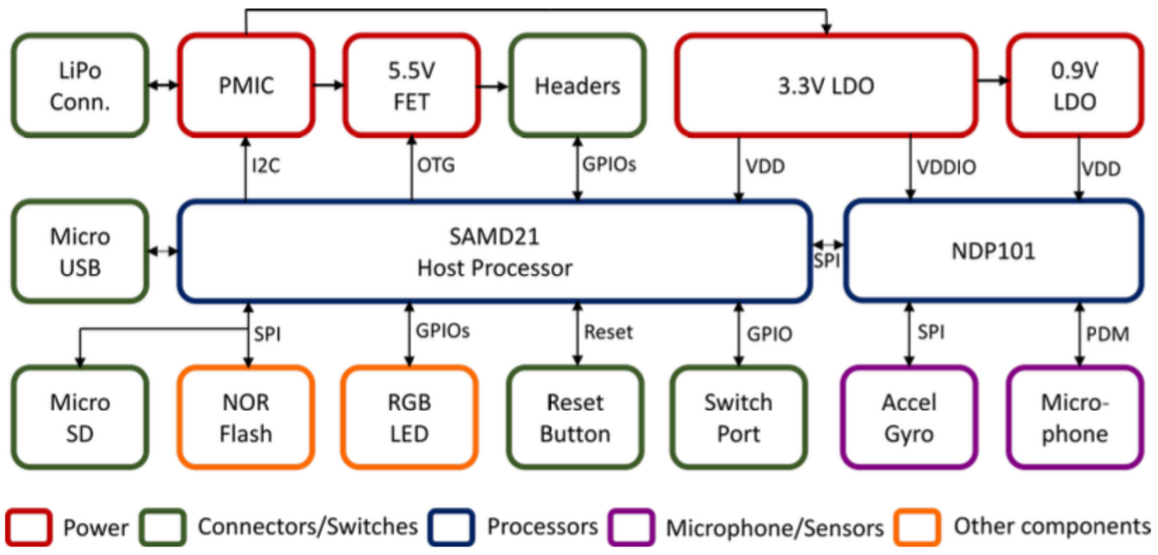

Figure 3.2: Design NDP101 with peripherals

Power metrics taking [?], considering that NDP101 is an always-on power consumption application it manages to use for audio/voice recognition applications 140μW and these results compared to other CPU will deliver 20 times more throughput and 200 times less energy per inference. The connection with another MCU is possible with SPI communication, however without the SDK provided is Syntiant, is difficult to program setup up it.

3.2 Syntiant Audio Block Processing

Edge Impulse reports that Syntiant Audio Block Processing is similar to MFE one[?]. This block objective is to process the raw data input in a input source extracting time and frequency features from this signal and in Syntiant case deffers a little because of a noise floor at the end of the computation. The block, which corresponds to the feature extractor, can be viewed as following: 1. Input - The

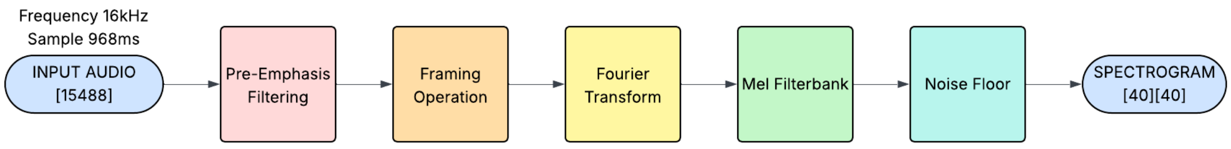

Figure 3.3: Syntiant Audio Block Processing

frequency of raw data input is at 16kHz and sampling 968ms, it corresponds in having 15488 raw data, with short values vecause the audio sound can go from -2^15 to $2^15 - 1$. These values come from an implicit ADC (Analog Digital Conversion), because Syntiant NDP101 has a dual PDM microphone input which interfaces with I2S interface multiplexed with PDM[?]. It stands for Pulse Density Modulation and it reduce the system into a single-bit digital one. This allows signal processing operations to be performed on the audio stream easily and then the PDM can be modified by the system in
2. Pre-emphasis filter - This is a high-pass filter that enhances high-frequency components, so the microphone will capture more low-frequency noise and increase high frequencies to make the speech clear. Before this is needed a audio normalization [-1, 1], generating floats values and then apply this high-filter:

$$y[n] = x[n] - \alpha \cdot x[n-1] \tag{3.1}$$

α consists in a coefficient of filter grade and a standard Syntaint Block set it at 0.96875
3. Framing - This audio is split and segmentate into small window called frames. Each frame has an

overlap time with each other and in Syntiant corresponds to 128 floats, considering the size of 512 floats and the stride, how much the start position will move of 384. In the last frame a part will overflow the initial buffer and in that case the void values are flatted to 0. Considering the input of 15488 samples in a 16kHz frequency:

$$number\,of\,frames = \frac{input\,size - frame\,size}{frame\,stride} + 1 = \frac{15488 - 512}{384} + 1 = 40 \tag{3.2}$$

For each one frame of the forty frames are computed:

3.1 Windowing - Before performing Fourier Transform, it's applied a windowing to reduce spectral leakage in integration. It is used the following sinusoidal function in each frame:

$$0.54 - 0.46 \cdot (\frac{2\pi k}{size - 1}) \tag{3.3}$$

The size will be 512 and k is an incremental value that goes from 0 up to 511 and k-window is multiplied to the k-position of the array. 3.2 Fast Fourier Transformation (FTT) - This function computes the complex frequency spectrum of a real-valued signal captured in time-domain. It is not required to compute all the DFT domain, because dealing with real value using Hermitian symmetry property with real values, it is require to compute only $\frac{N}{2} + 1$ unique complex outputs.

$$\begin{cases} X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{\frac{-2\pi ikn}{N}} & k = 0, 1, ..., \frac{N}{2} \\ X[l] = \overline{X[k]} & l = N - k \end{cases} \tag{3.4}$$

In this case x[n] is the input signal with n=0,...,N-1, x[n]$\in \mathbb{R}$, i is the imaginary unit, k is the incremental value and N is the length of the signal (512 values). 3.3 Spectrogram Population - Corresponds to a magnitude computation of the FFT output, obtaining the amplitude spectrum from the complex frequency-domain data. This computed the norm of the first half of the fourier transformation output and saves it in the corresponding size [40x256], the magnitude is computed as follows:

$$|X[k]| = \sqrt{(Re(X[k]))^2 + (Im(X[k]))^2},\ for\, k = 0, 1, ..., \frac{N}{2} - 1 \tag{3.5}$$

4. Mel-filterbank - After obtaining this matrix, the algorithm applies a mel-filterbank, which is a set of data based on human perception system via a triangular bandpass filter, making the system more sensitive to low frequencies. It is used the mel scale to obtain this phenomenon, because using a logarithmic approach allows a better sound recognition.
Internal in the software system is created a K+2 length filter called m, composed by linear spaced elements, with K the number of filters.

$$m[i] = M_{min} + i * \frac{M_{max} - M_{min}}{K + 1},\ i = 0, 1, ..., K - 1 \tag{3.6}$$

M_{min} and M_{max} corresponds to the conversion in mel-scale of the minimum and the maximum frequency. Syntiant as default has set the minimum to 0 and the maximum to $\frac{f_s}{2}$. After that it reconverts back in the frequency scale. The conversion formulas are for Syntiant:

$$m = 1127 \cdot log_{10}(1 + \frac{h}{700}) \quad h = 700 \cdot 10^{\frac{m}{1127}} - 1 \tag{3.7}$$

To, this scale is applied a triangular filter for each filter between 1 and K, using bins covering frequencies. The computation of the bins and of the triangular function is:

$$b_i = \lfloor \frac{2 \cdot f_i}{f_s} \cdot (N - 1) \rfloor \quad H_k[b] = \begin{cases} 0 & b < b_k - 1 or b > b_k + 1 \\ \frac{b - b_{k-1}}{b_k - b_{k-1}} & b_{k-1} \leq b \leq b_k \\ \frac{b_{k+1} - b}{b_{k+1} - b_k} & b_k \leq b \leq b_{k+1} \end{cases} \tag{3.8}$$

mel filterbank (pre-emphasis, framing, fft, mel scaling), mfe vs mfcc

3.3 Deep Learning Overview

3.4 Keyword Spotting Approaches

Classification, softmax

3.5 Speaker Verification Approaches

text-dependent vs independent sv, general methods not suitable for tinyml (gmm-ubm, i-vectors), d-vector concept, tinysv approach

4 Methodology

4.1 Hardware Pipeline

• Structure of the system (1 ESP32 + 2 Syntiant NDP101)
-¿ SPI communication NDP101 + Arduino MKRZero
-¿ SPI communication between ESP32 + NDP101

4.2 Software Pipeline

4.2.1 Signal Capture

Simulation (PortAudio), Real (Direct PDM sampling on NDP101) [Output: 15488]

4.2.2 MFE Block Generation

Simulation (Block explaining, validation with edge impulse generated spectrogram), real (Feature Extractor) [Output: 1600]

4.2.3 KWS Model

Input (1600) → FC256 (ReLU) → FC256 (ReLU) → FC256 (ReLU) → Output (4, Softmax)
Simulation (Weight extraction, target word, dataset), Real (model uploading and classification method)

4.2.4 SV Model

Input (40x40) → Conv+BatchNorm layers → 256-dimensional d-vector
Simulation (model-size, classification purpose, truncation explaination, verification methods (best-matching and mean-cosine)), Real (custom logic for verification)

Enrollment phase

Only for real logic, in simulation is not performed (various iteration of the process) [how to verify this? Model SV impossibility of uploading because of int-4 impossible deployment]

5 Software and Hardware Implementation

5.1 KWS Performance

Analysis using simulation data (confusion matrix)

5.2 SV Performance

Comparison between float32, int8 and int4 (int4 impossible to obtain good results using basic PQT, difficulties in deploying good QAT training, not true support by tensorflow and the fake quantization technique did not work)

5.3 System Integration

- Overall Power Consumption (ESP32 + 2 Syntiant NDP101)
- Memory Usage

6 Discussion

7 Conclusion & Future Work

7.1 Technical Limitations

- SDK Barriers (NDA limitations)
- Hardware Constraints (DNN-only architecture limiting SV capabilities)
- Quantization Challenges (Maintaining accuracy at 4-bit precision)

7.2 Design Trade-offs

- Accuracy vs Power (Best-matching vs mean-cosine approaches)
- Security vs User experience (Enrollment samples)
- Model Size vs Complexity (Feature extraction capabilities)

7.3 Key Contributions

- MFE/DNN Code - Full C implementation for Simulation in reproducing the behavior (the model is not quantized, so the behavior is ideal)
- Speaker Verification Distillated
- Quantization - Systematic approach to 8-bit quantization, not 4-bit (too experimental)

7.4 Future Directions

- If able to access to Syntiant, can deploy a compatible binary SV model
- Integration with existing system, that according to my system response will act as consequence
- Adaptive Training (The code is ready, but can't be tested properly)

Acknowledgements

*