POLITECNICO DI TORINO

Master's Degree in COURSE

Master's Degree Thesis

Real-time speech recognition using spiking neural networks

Supervisors

Prof. DI CARLO STEFANO

Prof. SAVINO ALESSANDRO

Dr. CARPEGNA ALESSIO

Candidate

BO WANG

10 2024

Summary

Spiking Neural Networks (SNN) have long been a popular research field which tries to combine neuroscience and machine learning to create models that most closely simulate the computational mechanisms of biological neurons. This research field, also known as neuromorphic computing, although very promising, is still not very mature in terms of real-world applications.

This thesis aims to cover this gap, exploring the application of SNN models to real-time speech recognition. The project covers the full data processing pipeline, from the raw audio signal, to the speech classification. It includes data encoding, which consists in converting sound to spikes, fine tuning of the SNN model, its offline training, and real-time inference using the trained network.
The work is implemented using low-cost commercial embedded boards. In particular, the audio sampling is performed using a STMicroelectronics® X-NUCLEO-CCA02M1™microphone. A first preprocessing is performed on a STMicroelectronics® NUCLEO-F446RE™microcontroller, directly connected to the microphone. The processed samples are than sent via USB to a Xilinx® PYNQ-Z2™embedded board. Here they are encoded into spikes and fed into the pre-trained SNN. This is executed using Python, exploiting the functionalities provided by the PYNQ project.
Different SNN models have been explored, searching for the best network, which resulted to be a recurrent network of Leaky Integrate and Fire (LIF) neurons, trained using Back-Propagation Through Time (BPTT) and the Surrogate Gradient technique. The network was implemented using PyTorch functionalities. Different encoding techniques have been considered as well, leading to an efficient encoding approach, based on 15 MEL distributed band-pass filters, which output was binarized to convert information in the spiking domain.

The overall system was first tested on the AudioMNIST dataset. This is composed of 30,000 recordings of spoken English numbers from 0 to 9. To the best of my knowledge this is the first time a SNN is applied to such dataset. The SNN was able to classify the input data with an accuracy of 96.49%. After this, the network was fine-tuned to recognize new samples, using the voice of two new users. In this

case the final accuracy on the new users settled around 93%. Finally, the network was deployed on the PYNQ-Z2™embedded board, performing inference on new incoming data, using the voice of the two new users. The accuracy was not affected, while the system was able to automatically detect the arrival of a spoken word and to classify it in real-time.

To sum up this work shows the potential of the application of SNNs in a real-world scenario, paving the way for a fast, flexible and efficient prototyping of neuromorphic models on resource-limited embedded board.

Acknowledgements

Firstly, I would like to express my heartfelt gratitude to my advisors, DI CARLO STEFANO and CARPEGNA ALESSIO. They gave me the opportunity to work on SNN research, which has truly excited me. Throughout the research process, they provided me with endless support and guidance, offering invaluable advice and helping me overcome numerous challenges.
I would also like to thank my wife,JiaJin Li, who has always supported me and even helped with recording and perfecting the dataset during the final adjustment phase of the project.
Once again, I am deeply grateful to everyone who has supported me in this work. Your support has meant the world to me.

Table of Contents

List of Tables

List of Figures

Acronyms

LSTM
: Long short-term memory

SNN
: Spiking neural network

LIF
: Non-recurrent network of non-adaptive LIF neurons trained with a surrogate gradient

adLIF
: Non-recurrent network of adaptive linear LIF neurons trained with a surrogate gradient

RLIF
: Recurrent network of non-adaptive LIF neurons trained with a surrogate gradient

RadLIF
: Recurrent network of adaptive linear LIF neurons trained with a surrogate gradient

FPGA
: Field Programmable Gate Array

SHD
: Spiking Heidelberg Digits

MFCC
: Mel Frequency Cepstral Coefficients

Chapter 1

INTRODUCTION

Spiking Neural Networks (SNNs) are models that mimic the mechanism of biological neurons for computation. By utilizing changes in membrane potential to trigger events and transmit information, they resemble brain-like computing. In research, SNNs have emerged as a new direction due to their higher efficiency and lower power consumption. However, because their algorithms and data processing methods are still being explored, they have not yet reached the level of industrial application. This thesis attempts to explore new encoding methods through speech recognition and deploy SNNs on a PYNQ board to bridge the gap between experimentation and practical use. In the following, we will discuss in detail the importance of speech encoding and why the PYNQ board was chosen for deployment.

In Spiking Neural Networks (SNNs), all input data should be encoded as spikes, using only 0s and 1s for representation. A 1 indicates that a spike is generated, while a 0 means no spike is generated. This means that the input data in SNNs carries temporal information, where spikes occurring at different times have different meanings. Moreover, the frequency of spikes represents different events, thereby also encoding spatial information. In speech processing, data has strong temporal and spatial relationships, making it a primary area of focus in SNN research.

ZYNQ is a new generation of All Programmable System on Chip (APSoC) launched by Xilinx, featuring highly flexible programmable logic (FPGA) that enables system optimization and differentiation, allowing the addition of custom peripherals and accelerators to adapt to a wide range of applications. PYNQ, short for Python + ZYNQ, refers to the Pythonization of certain ZYNQ functionalities, allowing direct use of Python libraries and FPGA hardware libraries for development. Since Python is currently the most widely used programming language for model development and PYNQ has excellent compatibility with Python, making it convenient for deployment and testing, this thesis uses PYNQ as the testing board for developing real-time speech recognition using spiking neural networks.

Chapter 2

Previous Work

2.1 Dataset Preparation

This thesis uses the Audio MNIST dataset, which consists of 30,000 audio samples of spoken digits (0-9) from 60 different speakers, all in mono.75% data was used as the training set, and 25% data was used as the test set. Specifically, the recordings of the first 45 individuals were used for training, while the recordings of the remaining 15 individuals were used for testing. The reason for this allocation is to evaluate the system's recognition accuracy for voices it has not encountered before.
Since the training set was recorded in a noise-free indoor environment, the model trained on this set would have relatively weak noise resistance. To solve this problem, this thesis enhanced the dataset by expanding it through methods such as adding noise, slowing down the audio, and altering the pitch.
This thesis also employs datasets traditionally used by SNN models. The SHD dataset, which is based on the HD dataset, is a spiking dataset that uses Lauscher encoding. Additionally, the HD dataset is used, which contains noise-free, pure high-definition audio recordings of the digits 0-9 in both English and German, with each digit repeated about 40 times for each speaker. Both the SHD and HD datasets are used to compare the results of the model training with those obtained from the AUDIO MNIST dataset.
Due to the limitations of the recording environment, recording devices, and the dataset of the training set, I used my system to record audio during the fine-tuning phase. I specifically recorded my voice and my wife's voice as a fine-tuning training set to adjust the model and improve recognition accuracy.

2.2 Hardware Preparation

The microphone used in this thesis is the X-NUCLEO-CCA02M1 MEMS digital microphone, which was utilized for fine-tuning and testing to meet the requirements of real-time system recognition.
For deploying the entire real-time speech recognition system, I chose to use PYNQ, a board based on ZYNQ. The PYNQ can execute Python which is so convenient for development of Python.
The system image file installed on the PYNQ is version v3.0.1.

2.3 Model Preparation

In traditional speech recognition tasks, LSTM is the most well-established and suitable model. As a comparison model, this thesis uses an LSTM with added self-attention layers to test the model's capabilities.

2.4 Encoding

In SNNs, encoding is a crucial aspect that has been under exploration. Currently, the mainstream encoding method is based on LAUSCHER's spike encoding, which uses an artificial model of the inner ear and parts of the ascending auditory pathway. The advantage of this encoding method is that it fully simulates the neurological characteristics of the human ear, making it seem like a natural match for SNNs, as both are inspired by biological features. However, its drawback is also evident—memory consumption might be huge for long audio files and each audio file, less than 2 seconds long, requires 10 seconds with 15 channels to encode. This is due to the characteristics of the model itself.

2.5 SNN

Currently, the popular types of SNNs include LIF, adLIF, RLIF, and RadLIF. LIF is the original model.

2.5.1 LIF

The Leaky Integrate-and-Fire (LIF) model is a simplified neuron model commonly used in neuroscience to simulate the spiking behavior of neurons. It is a key model in Spiking Neural Networks (SNNs). The LIF model simulates the membrane potential

changes of neurons through circuit-like behavior and describes how neurons generate spikes by receiving external input currents.[1]
The LIF model is the most classic and simplest spiking neural network model. It describes the changing process of a neuron's membrane potential. When the membrane potential exceeds a fixed threshold, it generates a spike. After the spike is fired, the membrane potential resets to the resting potential and then continues to accumulate, repeating this process.

The dynamics of the LIF model are described by the following equation[2]:

$$\tau \frac{du(t)}{dt} = -(u(t) - u_{\text{rest}}) + RI(t)$$

where:

- $u(t)$: membrane potential at time t; - u_{rest}: resting potential; - R: resistance; - $I(t)$: input current; - τ: membrane time constant, which controls the rate of change of the membrane potential.

2.5.2 adLIF

Although the LIF model is more widely used, its structure is too simple to reproduce the different firing patterns observed in biological neurons, such as adaptation, bursting, transient, and delayed firing. Therefore, the adLIF model was developed, which introduced a modern formulation where the recovery variable $w(t)$ is linearly coupled with the subthreshold membrane potential $u(t)$, and a mechanism is used to implement spike-triggered adaptation. The resulting more complex adaptive linear LIF (adLIF) neuron dynamics follow the differential equations[2]:

$$\tau_u \frac{du(t)}{dt} = -(u(t) - u_{\text{rest}}) - Rw(t) + RI(t) - \tau_u(\vartheta - u_r) \sum_f \delta(t - t_f)$$

$$\tau_w \frac{dw(t)}{dt} = -w(t) + a(u(t) - u_{\text{rest}}) + \tau_w b \sum \delta(t - t_f)$$

where:

- τ_u is the membrane time constant, determining how quickly the membrane potential $u(t)$ decays over time.
- τ_w is the adaptation time constant, controlling the dynamics of the adaptation current $w(t)$.
- $u(t)$ is the membrane potential at time t.

- u_{rest} is the resting membrane potential, the baseline potential when there is no input.
- R is the membrane resistance, influencing how the membrane potential reacts to input currents.
- $I(t)$ is the input current at time t, which drives the membrane potential.
- $w(t)$ is the adaptation variable (or current) that evolves over time and contributes to the neuron's adaptive behavior.
- a is the subthreshold adaptation parameter, controlling how $w(t)$ responds to the membrane potential $u(t)$ when no spike occurs.
- b is the spike-triggered adaptation parameter, controlling how much $w(t)$ increases immediately after a spike.
- ϑ is the firing threshold, the value of the membrane potential at which a spike is generated.
- u_r is the reset potential, the value to which the membrane potential is set after a spike.
- $\delta(t - t_f)$ is the Dirac delta function, representing the exact moment t_f when a spike occurs.
- t_f is the spike firing time, the moment when the membrane potential reaches ϑ and a spike is emitted.

2.5.3 RLIF

The RLIF (Recurrent Leaky Integrate-and-Fire) model is an enhancement of the LIF model and functions as a recurrent neural network. RLIF retains the basic structure of LIF neurons while enhancing its capability to process temporal data through recurrent connections. In the RLIF model, neurons have recurrent connections that allow them to handle sequential data. For example, the spike firing at the previous time step s_{t-1} can influence the membrane potential $u(t)$ at the current time step t. These recurrent connections improve the network's ability to model time-dependent inputs.

Training method: The RLIF model is trained using the surrogate gradient method. Due to the discontinuous nature of spike firing in spiking neural networks (i.e., the binary firing mechanism), traditional backpropagation algorithms cannot directly optimize the model. The surrogate gradient method employs differentiable

approximation functions to replace the spike firing function, enabling the network to be trained using gradient descent. [2] The equations is shown below

$$\tau \frac{du(t)}{dt} = -(u(t) - u_{\text{rest}}) + RI(t) + \sum_j W_{ij} s_j(t)$$

Each parameter in the equation has the following specific meaning:

- τ: Membrane time constant. This parameter controls how quickly the membrane potential $u(t)$ returns to the resting potential u_{rest}. A larger τ means the membrane potential changes more slowly.

- $u(t)$: Membrane potential at time t. This represents the voltage across the membrane of the neuron.

- u_{rest}: Resting potential. This is the default or resting membrane potential of the neuron when there is no input. It acts as a baseline voltage.

- R: Membrane resistance. This parameter influences how the input current $I(t)$ affects the membrane potential. A higher resistance R means that the same input current will cause a larger change in the membrane potential.

- $I(t)$: Input current at time t. This represents the external input to the neuron, which can come from sensory inputs or other neurons.

- W_{ij}: Synaptic weight between neuron i and neuron j. This parameter determines the strength of the connection from neuron j to neuron i. A higher weight W_{ij} results in a stronger influence from neuron j on neuron i.

- $s_j(t)$: Spike train from neuron j at time t. This is the binary value indicating whether neuron j has fired a spike at time t. A value of 1 indicates a spike, and a value of 0 indicates no spike.

It can be seen that, compared to the LIF model, the RLIF model only adds connections between different neurons. Although this increases the computational load, it significantly enhances the model's ability to capture information in time sequences.

2.5.4 RadLIF

RadLIF (Recurrent Adaptive Leaky Integrate-and-Fire) model is a combination of the RLIF model and the adLIF model. It inherits the recurrent feedback properties of the RLIF model and the adaptive characteristics of the adLIF model, allowing neurons to adjust their firing patterns according to the temporal characteristics of the input, while maintaining strong time-series processing capability. Below is a detailed description of RadLIF:

1. Core Characteristics of the RadLIF Model: - Recurrent Connections: In the RadLIF model, neurons have recurrent connections, enabling the output of a neuron to feed back into other neurons. This feedback mechanism allows the model to capture temporal dependencies in the input sequence.

Adaptive Mechanism: RadLIF introduces an adaptive mechanism where, after a neuron fires, the membrane potential recovery is controlled by an additional variable, causing the neuron's sensitivity to subsequent inputs to gradually decrease. This adaptive mechanism is achieved through the adaptive variable $w(t)$, which increases upon spike firing, thus adjusting the subsequent spike firing frequency.

2. Dynamical Equations of the RadLIF Model: The neuron dynamics of the RadLIF model are described by the following two main differential equations:

Membrane Potential Update Equation (combining recurrent and adaptive mechanisms):

$$\tau_u \frac{du(t)}{dt} = -(u(t) - u_{\text{rest}}) - Rw(t) + RI(t) + \sum_j W_{ij} s_j(t)$$

where:

τ_u is the membrane time constant; $u(t)$ is the membrane potential; u_{rest} is the resting potential; $w(t)$ is the adaptive variable; R is the resistance; $I(t)$ is the input current; W_{ij} is the weight matrix between neurons; $s_j(t)$ is the spike firing of other neurons at time t.

Adaptive Variable Update Equation (describing recovery after spike firing):

$$\tau_w \frac{dw(t)}{dt} = -w(t) + a(u(t) - u_{\text{rest}}) + \tau_w b \sum \delta(t - t_f)$$

where:

τ_w is the time constant of the adaptive variable; a and b are adaptive parameters; $\delta(t - t_f)$ is the Dirac delta function, representing spike firing at time t_f.

3. Training of RadLIF: The RadLIF model is trained using the surrogate gradient method. Due to the discontinuous nature of the spike firing process, standard backpropagation cannot be directly applied to spiking neural networks. The surrogate gradient method approximates the spike firing process as a differentiable function, allowing the backpropagation algorithm to train the RadLIF model. The thesis mentions that RadLIF uses the boxcar[2] surrogate gradient function, defined as:

$$\frac{\partial s(t)}{\partial u(t)} = \begin{cases} 0.5 & \text{if } |u(t) - \theta| \leq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

2.6 LSTM

LSTM (Long Short-Term Memory) is a special type of Recurrent Neural Network (RNN) designed to address the issues of vanishing and exploding gradients when dealing with long sequences. Compared to standard RNNs, LSTM introduces a

"memory cell" and three gating mechanisms (input gate, forget gate, and output gate) to better capture and retain long-range dependencies. Below is a detailed description of LSTM:

2.6.1 Memory Cell

The core of LSTM is a "memory cell" that is passed along through time, acting like a "highway" for information, allowing it to persist over multiple time steps without being easily altered. This memory cell is controlled by the three gating mechanisms, which decide when to add, remove, or output information. As a result, the memory cell can retain important information for extended periods without premature forgetting.

2.6.2 Gating Mechanisms

The gating mechanisms in the LSTM network are the key components, consisting of the forget gate, input gate, and output gate. Each gate uses a sigmoid activation function to compress input values between 0 and 1, acting as a switch to control the flow of information.

Forget Gate:

Function: Decides how much information in the memory cell should be forgotten. Working Principle: Receives the current input and the hidden state from the previous time step, generating a value between 0 and 1, where 0 means "forget everything" and 1 means "keep everything". The equation is:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

where$f_t$is the forget gate's output,$x_t$is the current input,$h_{t-1}$is the previous hidden state, and W_f and $b_f$are the weights and bias.

Input Gate:

Function: Decides how much of the new information should be added to the memory cell. Working Principle: The input gate works in two parts: one part decides how much new information to update, and the other part generates new candidate information. The equations are:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

for the input gate, and:

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

for the candidate information, where i_t is the input gate's output, and C_t is the new candidate information.

Output Gate:

Function: Controls how much information from the memory cell is output at the current time step. Working Principle: The output gate combines the current input and the hidden state to decide how much information from the memory cell will be output. The equation is:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

and the hidden state output is:

$$h_t = o_t \cdot \tanh(C_t)$$

where C_t is the updated memory cell state, and h_t is the final hidden state output.

2.6.3 Working Process

The working process of LSTM at each time step is as follows:

The forget gate decides which parts of the old information should be discarded. The input gate decides which new information should be added to the memory cell, and combined with the forget gate, it updates the memory cell. The output gate determines how much information from the memory cell is passed to the external environment.

2.6.4 Advantages

LSTM effectively mitigates the vanishing gradient problem in traditional RNNs when processing long sequences, as the memory cell can retain key information over time through the gating mechanisms. LSTM performs well in many applications, particularly in tasks that require handling long-term dependencies, such as natural language processing, speech recognition, and time series forecasting.

2.6.5 LSTM Summary

In summary, LSTM uses its memory cell and gating mechanisms to provide a significant advantage in modeling long-term dependencies while maintaining the RNN's ability to handle sequential data.

2.7 Hardware Description

The PYNQ-Z2 Development Board

The PYNQ-Z2 is a development board based on the Xilinx Zynq-7000 series System on Chip (SoC), designed to simplify the FPGA development process, especially for Python users. Here are its main features[3]:

Processor Architecture: The PYNQ-Z2 is equipped with the Xilinx Zynq-7000 series SoC, integrating a dual-core ARM Cortex-A9 processor and programmable logic (FPGA). The ARM processor handles operating systems and general software tasks, while the FPGA can be used to accelerate compute-intensive tasks[3].

FPGA Section: The FPGA logic in the Zynq-7000 SoC offers high flexibility, allowing users to design custom hardware circuits using hardware description languages (HDL) or high-level synthesis tools (like Vitis or Vivado). The PYNQ-Z2 particularly supports the PYNQ framework, which enables control and management of FPGA-accelerated tasks via Python, simplifying the development process.

Memory and Storage: The board has 512 MB of DDR3L SDRAM, shared between the ARM processor and FPGA. Additionally, it supports a microSD card, which is used to store the operating system (usually PetaLinux or Ubuntu) and user data.

Programmable Logic (PL) I/O: The PYNQ-Z2 is equipped with various input/output interfaces, including:

- Two Pmod connectors supporting multiple Pmod peripherals
- An Arduino connector compatible with Arduino Shields
- HDMI input and output for video processing applications
- VGA output interface
- Audio input/output interfaces (for audio processing)
- Multiple GPIO interfaces

Networking and Communication: The onboard Gigabit Ethernet interface supports fast network communication, making it suitable for applications requiring remote data transmission or cloud connectivity[3].

Operating System Support: The PYNQ-Z2 typically runs the PYNQ operating system, which is a customized system based on PetaLinux, specifically designed to simplify FPGA and Python integration. Users can easily write code in Python using Jupyter Notebooks and leverage libraries like NumPy or SciPy to control FPGA-accelerated hardware functions.

Power Supply: The PYNQ-Z2 can be powered via USB or with a 12V power adapter, providing flexibility to meet different power requirements.

Development Environment: Developers can use Xilinx's Vivado tools for FPGA development, or they can leverage the PYNQ environment to directly invoke FPGA acceleration features from Python code, significantly lowering the barrier to hardware programming.

Application Scenarios: The PYNQ-Z2 is widely used in embedded systems development, edge computing, machine learning acceleration, video and image

processing, and more. Due to its powerful ARM processor and FPGA acceleration capabilities, the PYNQ-Z2 excels in high-performance computing and real-time processing tasks.

The PYNQ-Z2 development board, with its simplified Python programming interface and customizable FPGA acceleration, is ideal for users who want to explore FPGA programming while reducing the complexity of hardware development.

2.8 Data Process

2.8.1 Lauscher

The Lauscher model applied in the Spiking Heidelberg Digits (SHD) dataset is a simulation of the human auditory system. Its primary function is to convert audio waveforms into spike trains, which can then be used for training spiking neural networks (SNNs). This conversion process involves three main steps:
Cochlear Model Simulation: First, the input audio signal is processed through a cochlear model. This model is based on the structure of the biological cochlea and uses shallow water wave equations to simulate how the basilar membrane responds to sound waves. The basilar membrane is a key structure in the ear, which vibrates differently depending on the frequency of the sound[4]
Hair Cell Model: After the cochlear processing, the audio signal is transformed into mechanical vibrations, which are then converted into electrical signals by a hair cell model. This step mimics how hair cells in the ear convert physical movements into bioelectric signals, similar to how the actual auditory system functions.[4]
Bushy Cells and LIF Neurons: In the final step, the electrical signals are further processed by bushy cells, which are auditory neurons. These cells use the Leaky Integrate-and-Fire (LIF) neuron model to convert the signals into spike trains. The LIF model captures how neurons accumulate charge over time and fire when a certain threshold is reached.[4]
This biologically-inspired process ensures that the generated spike trains have a high degree of physiological plausibility. In the SHD dataset, the audio recordings are from multiple speakers of both German and English, and after processing through the Lauscher model, each audio sample is converted into a spike train over 700 input channels. These spike trains can then be used to train and evaluate the performance of SNNs in tasks such as speech recognition.
This bio-inspired processing not only enhances the applicability of SNNs but also provides neuro scientific researchers with an audio dataset that is grounded in realistic physiological processes

2.8.2 mfcc

MFCC (Mel Frequency Cepstral Coefficients) is a widely used feature extraction method in speech and audio signal processing.[5] It mimics how the human ear perceives different frequencies and is particularly suited to converting speech signals into a compact, easily processed set of features. The computation of MFCC involves several detailed steps, each rooted in both physiological and mathematical properties, allowing the extraction of coefficients that reflect the characteristics of speech. **Detailed Steps in MFCC Calculation Pre-emphasis:** The first step in processing audio signals is pre-emphasis, where a high-pass filter is applied to the signal. This enhances the high-frequency components, which are naturally weaker in speech signals. The filter typically follows the equation:

$$y[n] = x[n] - \alpha \cdot x[n-1]$$

where $x[n]$ is the input signal, $y[n]$ is the output signal after pre-emphasis, and α is typically set to 0.95.[5] **Framing:** Since speech signals change over time, the signal is divided into small overlapping frames (typically 20-40 ms). This assumes that within each frame, the signal's frequency content remains stable. Common frame sizes are around 25 milliseconds. **Windowing:** To minimize signal discontinuities at the frame boundaries, a windowing function (usually a Hamming window) is applied. The Hamming window is defined as:

$$w[n] = 0.54 - 0.46 \cdot \cos\left(\frac{2\pi n}{N-1}\right)$$

where N is the window length, and n is the index of the sample within the frame[5]. **Fast Fourier Transform (FFT):** Each windowed frame undergoes an FFT, transforming it from the time domain to the frequency domain. The FFT provides the signal's spectrum, showing the magnitude of various frequency components. **Mel Filterbank:** After obtaining the spectrum, a set of Mel-scaled filters is applied. The Mel scale approximates the human ear's sensitivity to different frequencies, with more filters in the lower frequency range and fewer in the higher range. The Mel scale is defined as:

$$mel(f) = 2595 \cdot \log_{10}\left(1 + \frac{f}{700}\right)$$

These triangular filters capture the energy in specific frequency bands, producing a series of energy values for each filter[5]. **Logarithmic Energy:** The energy values from each filter are then converted to the logarithmic scale, which reflects the human ear's perception of loudness. This step helps emphasize the differences between low-energy and high-energy signals. **Discrete Cosine Transform (DCT):** The log-energy values are passed through a Discrete Cosine Transform (DCT) to remove

correlations between the filter coefficients and compress the information. The result is a set of Mel-frequency cepstral coefficients, which represent the signal's spectral properties. Usually, only the first 12-13 coefficients are kept as they capture the most relevant information. **Dynamic Features (Delta Coefficients):** In addition to static MFCC features, delta (first derivative) and delta-delta (second derivative) coefficients are often calculated to capture temporal changes in the speech signal. These dynamic features add information about how the speech characteristics evolve over time, which is crucial for tasks like speech recognition.
Advantages of MFCC:

- **Physiological relevance:** MFCC mimics the human ear's frequency response, especially the detailed processing in lower frequencies, making it sensitive to speech-specific features.
- **Efficiency:** By using a compact set of coefficients, MFCC reduces the dimensionality of the audio data, while retaining important information.
- **Widespread use:** MFCC is the most commonly used feature extraction technique in speech recognition, speaker identification, emotion analysis, and music classification.

2.9 Data Encoding

2.9.1 SNN Data Encoding

Spiking Neural Networks (SNNs) are a class of neural networks that more closely mimic the behavior of biological neurons.[6] Unlike traditional artificial neural networks (ANNs), which rely on continuous activations, SNNs process information as discrete events called spikes. Neurons in an SNN communicate by emitting spikes, a form of binary event, only when their membrane potential crosses a certain threshold. This leads to several different encoding schemes to represent information through these spikes. Here's a detailed overview of the major SNN encoding methods:
1. Rate Coding
Rate coding is the most straightforward and biologically inspired encoding mechanism used in SNNs. In this scheme, the information is encoded by the firing rate of neurons, meaning the number of spikes a neuron emits over a given time window represents the intensity of the stimulus. The higher the firing rate, the stronger the encoded signal[4].
Pros:

- It is biologically plausible and supported by experimental neuroscience, where

the firing rate of neurons in the brain is known to correlate with stimulus intensity.

- Simple and intuitive, making it widely used in initial SNN implementations.

Cons:

- Inefficient in terms of time and computational resources because it may require a large number of spikes to encode information accurately.
- Does not fully exploit the temporal dynamics of spiking neurons.

2. Temporal Coding
Temporal coding, also known as time-to-first-spike or latency coding, focuses on the precise timing of individual spikes, rather than the number of spikes. In this encoding scheme, the time delay between stimulus onset and the first spike carries the information. The sooner the neuron spikes after receiving input, the stronger the stimulus is perceived[4]. **Pros:**

- More efficient compared to rate coding as fewer spikes are needed, making it faster and more suitable for real-time applications.
- Leverages the precise timing of spikes, which is more aligned with the high temporal resolution of biological neurons.

Cons:

- Temporal coding can be more sensitive to noise and variability in spike timing.
- Requires more careful control over timing mechanisms in the network, making it harder to implement in hardware and software.

3. Population Coding
In population coding, information is distributed across a population of neurons rather than relying on a single neuron. Multiple neurons work together to encode the same stimulus, and each neuron in the population may fire at different times. The overall pattern of activity across the population is what represents the stimulus.
Pros:

- More robust to noise as the information is spread across multiple neurons, meaning errors in individual neurons are less impactful.
- Provides flexibility in encoding complex stimuli by using a collective representation.

Cons:

- Requires more neurons and potentially more computational resources.
- Extracting the population's overall signal can be computationally intensive.

4. Phase Coding
Phase coding is a specialized form of temporal coding where the phase of a spike relative to an oscillatory cycle is used to encode information. In this case, the timing of the spikes is interpreted in relation to a global rhythm or reference oscillation, and the phase at which a neuron fires can carry information.
Pros:

- Efficient in leveraging the oscillatory patterns observed in brain activity, such as theta and gamma oscillations.
- Can encode information more densely since each neuron's spikes can be interpreted within the context of the oscillation's phase.

Cons:

- Requires a reference oscillation, adding complexity to the system.
- Phase coding is less understood and less commonly implemented in computational models compared to rate and temporal coding.

5. Rank Order Coding
Rank order coding involves encoding information based on the order in which neurons fire. The first neuron to spike conveys the strongest signal, the second conveys a slightly weaker one, and so on. The relative rank of spike events is what encodes the stimulus.
Pros:

- Fast and efficient since it relies only on the first few spikes.
- Suitable for rapid, low-latency processing.

Cons:

- Highly dependent on precise spike timing, making it sensitive to noise.
- It can be difficult to implement in networks with a large number of neurons.

6. Spike-Timing-Dependent Plasticity (STDP) and Learning Mechanisms
SNNs often employ Spike-Timing-Dependent Plasticity (STDP) as a learning rule, which is a biologically plausible method that adjusts the strength of connections (synaptic weights) between neurons based on the timing of spikes.[7] The core principle behind STDP is that if a presynaptic neuron fires shortly before a postsynaptic neuron, the synapse is strengthened (known as Hebbian learning). Conversely, if the postsynaptic neuron fires first, the synapse is weakened.[8]
Pros:

- STDP allows for unsupervised learning based on spike timing, leading to adaptive neural networks that can self-organize.
- It is highly biologically plausible, aligning with experimental observations in the brain.

Cons:

- Learning can be slow and requires careful tuning of parameters.
- The biological complexity of STDP can make it challenging to implement effectively in large-scale SNNs.

Summary of SNN Encoding

SNNs offer a more biologically realistic alternative to traditional ANNs by modeling the discrete spike events of biological neurons. Different encoding schemes—rate coding, temporal coding, population coding, phase coding, and rank-order coding—each offer unique advantages and challenges depending on the task and the desired efficiency. Hybrid approaches and learning rules such as STDP can further enhance the capabilities of SNNs by making them more adaptive and suitable for dynamic, real-world tasks

This detailed exploration of encoding schemes highlights the flexibility and complexity of SNNs, providing a powerful foundation for tasks requiring energy efficiency, real-time processing, and biological plausibility.

In this thesis, two encoding methods are primarily used: temporal coding and population coding.

MFCC and MFCC rate encoding

The data obtained after MFCC processing can be directly used as input. The MFCC-encoded data is in floating-point format, which can be fed into an LSTM model to evaluate the model's performance.In this thesis, a new rate encoding method is also used, which combines MFCC with temporal coding combined with population coding.This encoding can be used for SNN models.

Chapter 3

Method

3.1 Dataset

In this thesis, two datasets are primarily used: the Audio MNIST dataset and the Heidelberg Digits dataset.

3.1.1 Audio Mnist

The Audio MNIST dataset is a digit recognition dataset based on speech, designed for tasks involving speech recognition or audio processing. Its primary goal is to recognize the digits 0 to 9 through audio input.
Dataset Content: The dataset consists of audio files that capture human speech of digits from 0 to 9. Audio Format: The audio samples are mono recordings at 48 kHz and are saved in .wav format. Sample Size: The dataset contains 30,000 samples, recorded by 60 speakers (20 females and 40 males), with each person contributing 500 samples.

3.1.2 Heidelberg Digits Dataset

Heidelberg Digits Dataset consists of approximately 10000 high-quality aligned studio recordings of spoken digits from 0 to 9 in both German and English language. Recordings exist of 12 distinct speakers two of which are only present in the test set[4].The sampling frequency of the HD dataset is also 48 kHz.The Spiking Heidelberg Digits Dataset is a dataset based on the HD dataset, encoded using Lausher encoding, and is primarily used in SNN models.

3.1.3 Use Method

By using traditional MFCC encoding, Lauscher encoding, and a new encoding method, the application of these in LSTM is compared to the accuracy of using LSTM without any encoding. Additionally, the application of Lauscher encoding and the new spiking encoding method in different SNN models is compared to the accuracy of models without any encoding.

3.2 LSTM Model Description

3.2.1 Model Initialization (___init___ Method)

The initialization method consists of input parameters, LSTM layers, multi-head self-attention mechanisms, fully connected layers, and dropout. Here are the key components:

Input Parameters

The input parameters are:

- Input dimension: input_d.In this thesis, input_d refers to the number of audio features after encoding, and after MFCC processing, it is 15.
- Hidden layer dimension: hidden_d.In this thesis, through experimentation, the hidden layer dimension was set to 64.
- Layer dimension: layer_d.This refers to the number of LSTM layers, which was set to 2 in this thesis based on experimentation.
- Output dimension: output_d.Output_d refers to the classification output. For the Audio MNIST dataset, output_d is 10, and for the HD dataset, output_d is 20.
- Batch size: batch_size.In this thesis, the batch size is set to 100.

LSTM Layer

The LSTM layer is bidirectional and the core part of the model. The LSTM operation can be described as:

$$h_t, c_t = \text{LSTM}(x_t, (h_{t-1}, c_{t-1}))$$

where:

- h_t represents the hidden state of the LSTM
- c_t represents the cell state
- x_t is the input data at time step t

The output shape from the LSTM layer is:

$$\text{output\_shape} = (\text{batch\_size}, \text{seq\_len}, 2 \times \text{hidden\_dim})$$

Since it is a bidirectional LSTM, the hidden dimension is doubled to 2×hidden_dim.[9]

Multi-Head Self-Attention Mechanism

The multi-head attention mechanism at each layer can be described by the following equation:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where Q is the query, K is the key, V is the value, and d_k is the dimensionality of the key.[10]
In the model, there are 4 attention heads. After the attention layer, residual connections and layer normalization are applied:

$$\text{output} = \text{LayerNorm}(\text{lstm\_out} + \text{attention\_out})$$

Fully Connected Layers and Dropout

The fully connected layers can be represented as:

$$y = W \cdot x + b$$

where:

- W is the weight matrix
- x is the input feature
- b is the bias term

Non-linearity is introduced through the GELU activation functions:

$$\text{GELU}(x) = 0.5x\left[1 + \tanh\left(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)\right)\right]$$

Dropout is applied as:

$$\text{Dropout}(x) = x \cdot \text{Bernoulli}(p)$$

where p is the probability of dropping a neuron.

3.2.2 Forward Pass (forward Method)

LSTM Layer

In the forward pass, the input x is passed through the LSTM layer, producing the output out. The hidden and cell states are initialized as zeros:

$$h_{l0} = 0, c_{l0} = 0$$

The LSTM layer operates as:

$$\text{out}, (h_{l0}, c_{l0}) = \text{LSTM}(x, (h_{l0}, c_{l0}))$$

The output is transposed to fit the attention layer input:

$$\text{out} = \text{out}.\mathit{transpose}(0,1)$$

Multi-Head Attention

After transposing, the multi-head attention mechanism processes the output. The attention calculation is performed as:

$$\text{attention\_out} = \text{Attention}(\text{out}, \text{out}, \text{out})$$

Residual connection and layer normalization follow:

$$\text{lstm\_out} = \text{LayerNorm}(\text{out} + \text{attention\_out})$$

Fully Connected Layers

The output from the attention layer is passed through fully connected layers. For example, the first fully connected layer can be represented as:

$$\text{out}_4 = \text{GELU}(\text{linear4}(\text{lstm\_out}))$$

Output

After passing through several fully connected layers, the final output is:

$$\text{out}_9 = \text{GELU}(\text{linear9}(\text{out}_8))$$

3.2.3 Overall Model Structure

The overall structure of the model can be summarized as:

LSTM → Multi-head Attention → Fully Connected Layers → Dropout → Output

Summary

This model combines the temporal modeling capabilities of bidirectional LSTM with the global feature extraction of multi-head self-attention. The multi-layer fully connected network processes the features and outputs the final classification or regression result.

3.3 SNN

In this thesis, the SNN model consists of three layers: two layers as hidden layers of the spiking neural network, and one readout layer. In the hidden layers, a comparison was made between two LIFLayer, two adLIFLayer, two RLIFLayer, and two RadLIFLayer.

3.3.1 LIFlayer

LIFLayer (Leaky Integrate-and-Fire Layer): Description: LIF neurons are a type of Leaky Integrate-and-Fire neuron, which is the most basic spiking neuron model. It does not have adaptive properties or intra-layer recurrent connections.[2]
Parameters:

- **input_size:** Number of input features.In this thesis, for the SHD dataset, the input size is 700, while for the Audio MNIST dataset, it is 15.
- **hidden_size:** Number of output neurons.In this thesis, the hidden size is set to 128 for all datasets.
- **batch_size:** Batch size of the input.In this thesis, the batch size is set to 100 for all datasets.

- **threshold:** Fixed spiking threshold of the neurons.In all the SNN models, the threshold is set to 1.0.
- **dropout:** Dropout coefficient, used to prevent overfitting (dropout is set to 0.1).
- **normalization:** Batch normalization or layer normalization to standardize inputs.
- **use_bias:** Whether to use trainable bias.
- **bidirectional:** Whether to use bidirectional LIF neurons, processing the sequence in both forward and backward directions.

Working Principle:

1. The input undergoes a linear transformation before being passed to the membrane potential update of the neurons.
2. The potential is updated according to the dynamic equation of the LIF model, and the spikes are computed using a surrogate gradient to handle the non-differentiability of the spiking function.
3. After each spike, the membrane potential resets to zero, generating the spiking output.
4. Finally, dropout with a probability of 0.1 is applied to randomly discard some of the data to prevent overfitting.

membrane potential computation:
The formula for the membrane potential is:

$$u_t = \alpha \cdot (u_t - s_t) + (1 - \alpha) \cdot W_x[:, t, :]$$

where:

- u_t: Membrane potential at the current time step t, representing the neuron's voltage state at time t.
- α: A decay coefficient that controls the leak of the membrane potential. Its value typically ranges between 0 and 1. A larger α means the membrane potential decays more slowly, while a smaller α indicates faster decay. Here, α is a random number within the range $\exp\left(\frac{-1}{5}\right)$ and $\exp\left(\frac{-1}{25}\right)$.
- s_t: The spike activity at the previous time step $t-1$. If a spike was emitted in the previous step, s_t is usually 1; otherwise, it is 0. This value modulates the membrane potential, as the potential resets after a spike.

- $W_x[:, t, :]$: The input received by the neuron at the current time step t. This can be an input signal from the external environment or input from other neurons within the network.
- $1 - \alpha$: Balances the influence of the input on the membrane potential. The larger $1 - \alpha$ is, the greater the contribution of the input to the membrane potential.

3.3.2 adLIFLayer (Adaptive Leaky Integrate-and-Fire Layer):

Description: The adLIF neuron is an adaptive version of the LIF neuron. It adds an adaptive potential (slow potential) to simulate the fatigue effect observed in biological neurons.[2] **Parameters:** In addition to the parameters similar to those of `LIFLayer`, the following parameters are added:

- **a:** Adaptive parameter controlling the strength of the adaptive current.
- **b:** Adaptive offset that influences the dynamics of the adaptive current.
- **alpha:** Controls the decay rate of the membrane potential.
- **beta:** Controls the decay rate of the adaptive potential.

Working Principle:

1. The adLIF neuron computes the membrane potential based on the input current and gradually adds an adaptive potential over time, simulating the delayed response of neurons.
2. When the neuron spikes, the adaptive potential increases, which reduces the probability of subsequent spikes.
3. This adaptive mechanism helps the model capture dynamic changes in long-time sequences.

membrane potential computation:
The update formula for the adaptive variable of the cell is:

$$w_t = \beta \cdot w_t + a \cdot u_t + b \cdot s_t$$

where:

- w_t: Adaptive variable, representing the neuron's adaptive current. This variable is used to simulate the adaptive behavior of the neuron, such as reduced responsiveness to inputs after repeated activations.

- β: The decay coefficient of the adaptive variable, controlling the gradual decay of the adaptive current. Similar to the membrane potential time constant τ_w, a larger β value results in slower decay.Here, β is a random number within the range $\exp\left(\frac{-1}{30}\right)$ and $\exp\left(\frac{-1}{120}\right)$.
- a: Subthreshold adaptation coefficient, describing the influence of the membrane potential on the adaptive current. If u_t is high, the adaptive variable w_t increases more rapidly, slowing down future increases in the membrane potential.It is a random number within the range $(-1.0, 1.0)$.
- b: Spike-triggered adaptation coefficient, controlling the influence of the spike activity s_t on the adaptive variable w_t. When a neuron spikes in the previous time step ($s_t = 1$), the adaptive variable w_t increases, inhibiting future spikes. It is a random number within the range $(0.0, 2.0)$.
- u_t: The membrane potential at the current time step, which influences the adaptive variable w_t, adjusting future adaptive current.
- s_t: Spike activity at the previous time step. $s_t = 1$ indicates that a spike occurred in the previous time step, while $s_t = 0$ indicates no spike.

The membrane potential update formula is:

$$u_t = \alpha \cdot (u_t - s_t) + (1 - \alpha) \cdot (W_x[:, t, :] - w_t)$$

where:

- u_t: The membrane potential at the current time step, representing the neuron's voltage state. The membrane potential is the basis for spike generation, and when it exceeds the threshold, the neuron fires a spike.
- α: The decay coefficient, controlling the leakage rate of the membrane potential. A larger α value means slower decay of the membrane potential, while a smaller α value means faster decay.Here, α is a random number within the range $\exp\left(\frac{-1}{5}\right)$ and $\exp\left(\frac{-1}{25}\right)$.
- s_t: The spike activity at the previous time step. If a spike occurred in the previous time step ($s_t = 1$), the membrane potential u_t decreases.
- $W_x[:, t, :]$: Represents the input signal at the current time step t (which can be external input or input from other neurons).
- w_t: The adaptive variable, which slows down the growth of the membrane potential. If w_t increases, it indicates that the neuron becomes more inhibited, reducing future membrane potential growth.

3.3.3 RLIFLayer (Recurrent Leaky Integrate-and-Fire Layer):

Description: RLIF neurons are LIF neurons with recurrent connections. Neurons within each layer have feedback connections, allowing them to process recursive information from the time series.[2] **Parameters:** In addition to the parameters similar to those of `LIFLayer`, the following parameter is added:

- **V:** Recurrent matrix within the layer (recurrent weights connecting neurons within the same layer).

Working Principle:

1. Beyond the standard LIF dynamics, RLIF neurons use the recurrent matrix V to feed the spiking activity of each time step back to other neurons within the same layer, capturing more complex temporal dependencies.
2. The diagonal elements of the recurrent matrix V are set to zero to avoid self-feedback of individual neurons.
3. This recurrent structure enhances the model's ability to handle long-term dependencies.

membrane potential computation:
The linear layer representing the recurrent connection weight matrix:

```
self.V = nn.Linear(self.hidden_size, self.hidden_size, bias=False)
```

This is a linear layer (fully connected layer) used to represent the recurrent connection weight matrix between neurons. The input and output dimensions of this layer are both `self.hidden_size`, meaning it is a weight matrix used for interactions between neurons.
Setting the diagonal elements of the weight matrix to 0:

```
V = self.V.weight.clone().fill_diagonal_(0)
```

The diagonal elements of the weight matrix are set to 0. This is done to prevent neurons from having direct feedback to themselves (i.e., avoiding self-feedback), allowing only interactions between different neurons.
The membrane potential update formula:

$$u_t = \alpha \cdot (u_t - s_t) + (1 - \alpha) \cdot (Wx[:, t, :] + \texttt{torch.matmul}(s_t, V))$$

Where:

- u_t: The membrane potential at the current time step t, representing the neuron's voltage state.

- α: Decay coefficient, controlling the leak rate of the membrane potential. A larger α means slower decay of the membrane potential, while a smaller α means faster decay.
- $u_t - s_t$: Represents the current membrane potential u_t minus the spike activity from the previous time step s_t. This means that if a spike was emitted in the previous time step $s_t = 1$, the current membrane potential will decrease accordingly.
- $(1 - \alpha)$: This is a compensatory term, balancing the influence of input and feedback.
- $Wx[:, t, :]$: Represents the input at time step t, which is typically the input received by the neuron from external sources or other neurons.
- `torch.matmul`(s_t, V): Represents the multiplication of the spike activity from the previous time step s_t with the recurrent connection weight matrix V. This step introduces recurrent feedback, meaning the activity of other neurons from the previous time step influences the current membrane potential through the weight matrix V.

3.3.4 RadLIFLayer (Recurrent Adaptive Leaky Integrate-and-Fire Layer):

Description: RadLIF neurons are adaptive recurrent LIF neurons. They combine adaptivity (like in `adLIF`) with recurrent connections (like in `RLIF`), enabling them to handle long-term dependencies and simulate neuronal fatigue effects.[2]
Parameters: Similar to `RLIFLayer`, with the addition of adaptive parameters similar to `adLIFLayer`.
Working Principle:

1. RadLIF neurons combine intra-layer recurrent connections with an adaptive potential. This allows the neurons to adjust their membrane potential not only based on past inputs but also based on their spiking history, modulating the adaptive current accordingly.
2. The recurrent connections provide temporal dependency processing within the layer, while the adaptive current reduces the likelihood of continuous spiking, improving the model's ability to capture long-term dependencies.

membrane potential computation:
The formulas and explanations are as follows:

$$\alpha = \text{torch.clamp(self.alpha, min=self.alpha\_lim[0], max=self.alpha\_lim[1])}$$

α is the decay coefficient of the membrane potential, used to control the leak rate of the membrane potential over time. The range is $[\exp(-1/5), \exp(-1/25)]$.

$$\beta = \text{torch.clamp(self.beta, min=self.beta\_lim[0], max=self.beta\_lim[1])}$$

β is the decay coefficient of the adaptive variable, used to control the decay speed of the adaptive current. The range is $[\exp(-1/30), \exp(-1/120)]$.

$$a = \text{torch.clamp(self.a, min=self.a\_lim[0], max=self.a\_lim[1])}$$

a is the subthreshold adaptation coefficient, which controls the response of the adaptive variable w_t to the membrane potential u_t. The range is $[-1.0, 1.0]$.

$$b = \text{torch.clamp(self.b, min=self.b\_lim[0], max=self.b\_lim[1])}$$

b is the spike-triggered adaptation coefficient, which controls how the adaptive variable w_t changes after a spike is emitted. The range is $[0.0, 2.0]$.

$$V = \text{self.V.weight.clone().fill\_diagonal\_(0)}$$

V is the weight matrix in the recurrent neural network, which describes the connection strength between neurons.

$$w_t = \beta \cdot w_t + a \cdot u_t + b \cdot s_t$$

The adaptive variable w_t is updated based on the previous adaptive variable, the adaptation coefficient a, the membrane potential u_t, and the spike activity s_t.
$\beta \cdot w_t$: The decay of the adaptive variable, controlling its natural decay over time.
$a \cdot u_t$: The influence of the membrane potential on the adaptive variable, meaning that if the membrane potential is high, the adaptive variable accumulates more quickly.
$b \cdot s_t$: The influence of spike activity on the adaptive variable, meaning that after a spike is emitted, the adaptive variable increases, inhibiting subsequent inputs.

$$u_t = \alpha \cdot (u_t - s_t) + (1 - \alpha) \cdot (Wx[:, t, :] + \text{torch.matmul}(s_t, V) - w_t)$$

The membrane potential update formula:
$\alpha \cdot (u_t - s_t)$: This represents the decay of the membrane potential. If a spike was emitted in the previous time step ($s_t = 1$), the membrane potential will decrease.
$(1 - \alpha) \cdot (Wx[:, t, :] + \text{torch.matmul}(s_t, V) - w_t)$: This term represents the influence of the input signal and recurrent feedback.
$Wx[:, t, :]$: Represents the input signal at the current time step.
$\text{torch.matmul}(s_t, V)$: Represents the influence of the spike activity from the previous time step through recurrent connections on the current membrane potential.
$-w_t$: Represents the inhibitory effect of the adaptive variable on the membrane potential, preventing the membrane potential from increasing too quickly.

3.3.5 ReadoutLayer:

Description: The readout layer is a non-spiking layer used to output the accumulated membrane potential. This layer does not generate spikes; instead, it accumulates the input signal and converts the membrane potential into a probability distribution using softmax.
Parameters: Similar to `LIFLayer`, but without the spiking mechanism.
Working Principle:

1. The input data undergoes a linear transformation and is then accumulated as membrane potential without generating spikes.

2. Finally, the membrane potential is converted into an output probability distribution using softmax, making it suitable for classification tasks.

3. The readout layer is typically used as the final layer in the network, outputting class labels or continuous values.

Chapter 4

Audio Process

The primary difference between SNN and ANN lies in the use of spike encoding, which means the input data can only be 0 or 1. Compared to floating-point data, this holds significantly less information, making the encoding issue a critical aspect in SNNs. In previous studies, the most commonly used method is BM decompositions converted to phase-coded spikes using a transmitter-pool-based hair cell (HC) model.[7] However, this model is computationally slow, and when tested on the MNIST AUDIO dataset with this encoding method, the accuracy only reached around 68%, which is unacceptable for a real-time speech recognition system. Therefore, this thesis attempts to explore alternative encoding methods to improve the overall usability of the system.

4.1 Data Augmentation

In the model trainning phase, three encoding sets were used: 1. SHD dataset, 2. Audio MNIST dataset, and 3. Audio MNIST Proc dataset. The SHD and Audio MNIST datasets have already been introduced in previous sections. The third dataset, Audio MNIST Proc, is an extended version of the Audio MNIST dataset. Since the original Audio MNIST dataset only contains recordings from 60 speakers and each speaker recorde 50 times for each number, to increase diversity, the recordings from Audio MNIST were modified in six ways: speeding up, slowing down, adding noise, raising the pitch, lowering the pitch, and amplifying the sound, each generating a new set of recordings. As a result, the Audio MNIST Proc dataset contains a total of 210,000 recordings (training + testing). This approach significantly enhances the diversity of the data.

4.2 New Spiking Encoding

After looking at the result of accuracy of SHD and MNIST, we know that the mfcc is the very great method for audio encoding. However, the general mfcc encoding is the float type encoding which can not be used in spike network.In order to turn mfcc encoding to spike network, this thesis create a method for this functin. The fomula is show below.

$$a_i / \max\{a_1, a_2, \ldots, a_n\} < 0.975$$

a_i is a element of the audio encoding by mfcc, when this ratio is lower than 0.975, the spiking will be encoded to 0, otherwise 1. The 0.975 is the test value that is the best one to imporve accuracy.

4.3 The Final Result

Table 4.1: Accuracy Comparison for SHD and MNIST Audio Encoding Methods

Method	**SHD Accuracy (%)**	**MNIST Audio Accuracy (%)**	**MNIST proc audio Accuracy**
LSTM Original Audio (MFCC Encoding)	93.50	98.40	97.34
LSTM New Method Encoding	85.32	95.56	94.42
Lauscher (RLIF)	85.41	73.36	68.56
Lauscher (LIF)	82.60	68.77	67.01
Lauscher (adLIF)	91.60	71.87	65.77
Lauscher (RadLIF)	90.46	74.29	66.12
New Method Encoding(RLIF)	88.72	96.49	95.18
New Method Encoding(LIF)	82.40	93.22	92.38
New Method Encoding(adLIF)	88.54	91.58	93.6
New Method Encoding(RadLIF)	89.50	93.33	94.53

Conclusions

The following observations can be made:

- The LSTM using MFCC encoding achieved the highest accuracy across the three datasets.
- The accuracy of the LSTM model using spike encoding based on MFCC in the SHD dataset showed a significant difference compared to the LSTM using pure MFCC encoding, with the former being about 10% lower. However, in the MNIST audio and MNIST audio proc datasets, the LSTM model with MFCC-based spike encoding performed only 3% worse than the LSTM with traditional MFCC encoding.
- The SNN model using lauscher encoding performed well in the SHD dataset but poorly in the MNIST audio and MNIST audio proc datasets.
- In the new spike encoding based on MFCC, the SNN model in the SHD dataset performed similarly to the SNN model with lauscher encoding, but showed excellent performance in the MNIST audio and MNIST audio proc datasets, with an average accuracy improvement of 20%.
- For the SHD dataset, in the SNN model using lauscher encoding, the best performers were adLIF and RadLIF, both achieving accuracies above 90%. In the new spike encoding based on MFCC, the best-performing models were RLIF, adLIF, and RadLIF, with little difference among the three.
- In the MNIST audio and MNIST audio proc datasets, the SNN model with lauscher encoding performed poorly, with accuracies not exceeding 75%.
- In the MNIST audio and MNIST audio proc datasets, the highest-performing models using the new spike encoding based on MFCC were RLIF and RadLIF.
- Although the MNIST audio proc dataset is an extended version of the MNIST audio dataset, the experiments revealed that the increased data did not have a positive effect on model training. In fact, accuracy decreased in all models compared to the MNIST audio dataset.

When comparing the results of various models on the SHD dataset and MNIST datasets, the following conclusions can be drawn:

1. Compared to SNN models, LSTM models using floating-point data performed the best. If spike encoding was used, the accuracy was lower than with floating-point encoding.

2. In multi-class classification (20 classes) with small datasets, the lauscher encoding slightly outperformed MFCC-based spike encoding.

3. In fewer-class classification (10 classes) with large datasets, the SNN models using lauscher encoding performed poorly.

4. In fewer-class classification (10 classes) with large datasets, the SNN models using MFCC-based spike encoding performed well. In fact, the highest-performing RLIF model had an accuracy only 2% lower than the LSTM model.

5. Data augmentation through basic transformations may yield worse results than training with the original data.

6. Comparing the accuracy of LSTM models using MFCC-based spike encoding to SNN models using the same encoding, the SNN models performed better on the SHD dataset than LSTM models.

Chapter 5

Hardware Configuration

By comparing the model results, it was found that using the RLIF model with MFCC-based spike encoding achieves the highest accuracy. Therefore, it was decided to use the RLIF model and MFCC-based spike encoding in the system.

5.1 Mic

The microphone used is the X-NUCLEO-CCA02M1, which is an STM32 Nucleo expansion board designed for high-performance MEMS microphone arrays. This expansion board is equipped with the MP34DT01-M digital microphone.

5.1.1 X-NUCLEO-CCA02M1 Features

1. **Multi-channel audio acquisition:** The board integrates two MP34DT01-M digital MEMS microphones, supporting up to two microphone inputs, enabling high-fidelity audio capture, making it particularly suitable for voice processing tasks.

2. **Supported audio sampling rates:** It supports audio sampling rates ranging from 8 kHz to 96 kHz, providing great flexibility for various audio processing requirements.

3. **Audio processing:** Combined with the STM32 microcontroller, it allows efficient audio signal processing by converting PDM (Pulse Density Modulation) to PCM (Pulse Code Modulation).

4. **Compatibility:** Compatible with STM32 Nucleo development boards, it connects via Arduino-compatible expansion slots to the Nucleo board.

5. **STM32Cube software support:** The expansion board is fully compatible with the STM32Cube software ecosystem. It can leverage the STM32Cube-generated HAL library, PDM2PCM software library, and X-CUBE-MEMSMIC1 expansion package, simplifying the development process.

5.1.2 MICs Description

- **MEMS microphones:** The board features two MP34DT01-M digital MEMS microphones, which output audio signals via Pulse Density Modulation (PDM).
- **Expansion interface:** The board includes standard Arduino expansion slots, making it easy to connect to STM32 Nucleo development boards such as the NUCLEO-F401RE or NUCLEO-L476RG.
- **PDM input interface:** The board has two PDM input interfaces, supporting audio data acquisition from one or two microphones.
- **Power consumption:** The board is designed for low power consumption, making it suitable for battery-powered portable devices.

The sound processing is handled by the NUCLEO-F446RE development board, which is part of the STM32 Nucleo series and based on the STM32F446RE microcontroller. It features high performance and low power consumption.

5.2 Stm32F446RE Overview

5.2.1 Microcontroller: STM32F446RE

- **Core:** Cortex-M4 32-bit RISC core, with a maximum frequency of 180 MHz, supporting single-precision floating-point unit (FPU).
- **Memory:** 512 KB Flash memory and 128 KB SRAM.[11]
- **Operating Voltage:** 3.3V core with 5V-tolerant pin inputs.
- **GPIO Pins:** 50 general-purpose input/output (GPIO) pins, supporting various functions such as external interrupts, ADC, PWM, and digital I/O.[11]

5.2.2 Power Supply

- **USB Power:** Can be powered via USB connection.
- **External Power Supply:** Supports external power input (7V-12V), through the Vin pin or the onboard 5V power socket.

5.2.3 Interfaces

- **USB:** Supports USB OTG, which can function as both device and host, suitable for USB communication and data transfer.

- **Serial Communication:**

  - **USART/UART:** Supports multiple UART and USART interfaces for serial communication.

  - **I2C:** Up to three I2C interfaces, supporting multi-master and multi-slave communication.

  - **SPI:** Supports up to four SPI interfaces for high-speed data transfer.

  - **CAN Bus:** Supports CAN bus, used in industrial control and automotive applications.

- **ADC and DAC:**

  - **ADC:** 16-bit Analog-to-Digital Converter (ADC), supporting multi-channel sampling.

  - **DAC:** Supports 12-bit Digital-to-Analog Converter (DAC), for generating analog signals.

5.3 Audio Acquisition

Audio Acquisition work flow is shown below

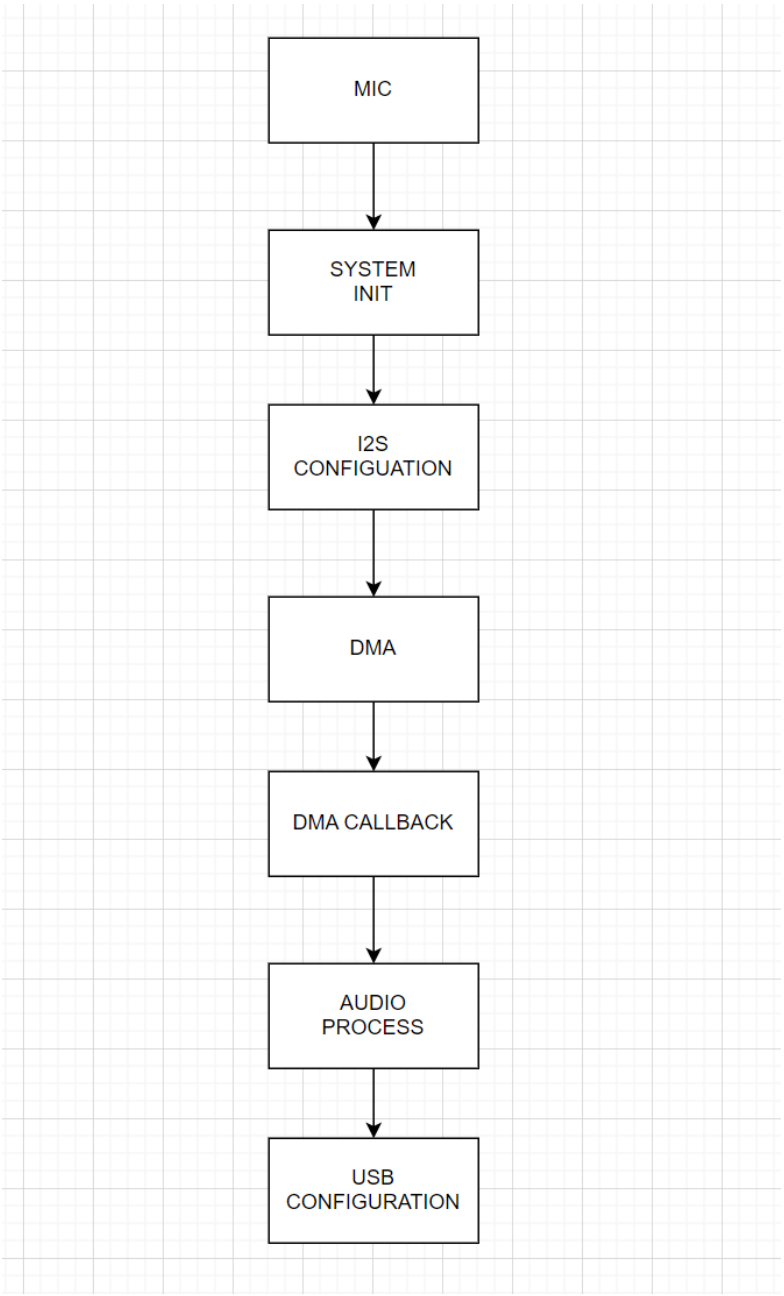

Figure 5.1: Audio Acquisition

5.3.1 MIC Configuration

The audio acquisition uses the MEMS microphone (MEM1) from the X-NUCLEO-CCA02M1. The sampling frequency is set to 48 kHz, and the audio is configured

to be mono with a volume level of 64U.

5.3.2 System Init

AHB (Advanced High-performance Bus) is part of the ARM AMBA (Advanced Microcontroller Bus Architecture) specification and is designed to handle high-speed data transfers between various system components. It is typically used as the main system bus in microcontroller systems, enabling communication between the processor, memory, and peripherals.
Here is a more detailed explanation of AHB:

Key Features of AHB

High Performance

- AHB is designed to support high data throughput, making it suitable for handling communication between high-speed peripherals and the CPU.
- It allows single-cycle data transfer, which improves performance, especially for operations that require frequent memory access.

Bus Structure

- AHB operates on a master-slave architecture, where multiple master devices (such as the CPU, DMA, or other controllers) can access slave devices (such as memory, peripherals, or I/O) through a centralized bus.
- The bus arbiter controls which master device gets access to the bus, ensuring smooth communication between components.

Burst Transfers

- AHB supports burst transfers, where multiple data items can be transferred consecutively without the need to reinitiate the bus transfer for each data item. This reduces overhead and increases efficiency.

Pipelining

- AHB supports pipelined operations, allowing address and data phases to overlap. This enables faster data transfers since the next transaction can start before the current one finishes.

Clock Synchronization

- The AHB clock is typically synchronized with the system clock (SYSCLK), which is why the AHB clock is often equal to the system clock frequency. In your configuration, the AHB bus clock operates at 180 MHz, the same as the system clock.

AHB-Lite

- Many microcontrollers, including STM32, use a simplified version of AHB called AHB-Lite, which supports a single master on the bus, reducing complexity for embedded systems that don't require multiple masters.

The system crystal oscillator is set to HSE, with a base frequency of 8 MHz, a division factor of 8, and a multiplication factor of 360. The main system clock is divided by 2, resulting in a final system frequency of 180 MHz. The PLLCLK is set as the main system clock source. The AHB bus clock is the same as the system clock. The APB1 bus clock is 180/4 = 45 MHz, and the APB2 bus clock is 180/2 = 90 MHz.

APB1

APB1 (Advanced Peripheral Bus 1) is an internal bus in the STM32 microcontroller and is part of its bus matrix, used to connect low-speed peripherals. It plays an important role in the clock tree of the STM32 microcontroller, providing clock signals and data transmission channels for the peripherals mounted on it. APB1 is mainly used to connect low-speed peripherals such as timers, UART, I2C, and SPI. Compared to APB2 (another peripheral bus for high-speed peripherals), APB1 typically runs at a lower clock frequency.

Working Principle of APB1 In the STM32 architecture, the system clock is distributed through several different buses, including AHB (Advanced High-performance Bus), APB1, and APB2. APB1 is a subordinate bus to AHB, providing clock and data paths for lower-speed peripherals.

APB1 Clock Source and Frequency The clock frequency of APB1 is derived by dividing the system clock (SYSCLK). The specific division factor is set by the APB1 prescaler. Common APB1 prescaler values include 1, 2, 4, 8, and 16. Through this prescaler, the clock frequency of APB1 is typically lower than that of APB2.

System Clock Configuration In our system, the system clock (SYSCLK) is 180 MHz, and the APB1 prescaler is set to 4, then the APB1 clock frequency is:

$$f_{\text{APB1}} = \frac{f_{\text{SYSCLK}}}{4} = \frac{180\,\text{MHz}}{4} = 45\,\text{MHz}$$

The external crystal oscillator frequency of the system is 16 MHz. The sysTick is set to 16 kHz, so an interrupt is generated every 1 ms.

5.3.3 I2S Configuation

I2S (Inter-IC Sound) is a serial communication protocol developed by Philips (now NXP) for transmitting digital audio data. [12]It is specifically designed for transferring audio data between integrated circuits, such as microcontrollers, audio codecs, digital signal processors, DACs, and ADCs. I2S is widely used in audio equipment, embedded systems, sound cards, Bluetooth audio devices, and more. I2S is designed to be simple and efficient, supporting synchronous transmission of high-quality audio, including stereo and multichannel audio.

1. Basic Concepts and Signal Lines of I2S I2S communication transmits audio data through serial data lines, primarily including the following signal lines:

- **SCK (Serial Clock):** Also known as BCLK (Bit Clock), used to synchronize the transmission of data. Each bit of audio data is transmitted in one clock cycle of SCK. It is generated by the master device and transmitted to the slave device. The master can be a microcontroller, processor, or audio controller, and the slave can be an audio codec, DAC, ADC, etc.
- **WS (Word Select):** Also called LRCLK (Left/Right Clock) or frame sync signal. It distinguishes between left and right channel data. When WS is low, left channel data is transmitted; when WS is high, right channel data is transmitted.[12] The frequency of WS is the audio sampling rate, representing the start of each frame of audio data (left and right channels).
- **SD (Serial Data):** Audio data is transmitted via the SD line, which carries the actual audio sample data. Data is typically transmitted MSB (Most Significant Bit) first, with the data from each channel (left and right) sent in sequence through this line.
- **MCLK (Master Clock) (Optional):** Used in some systems, especially audio codecs. MCLK provides a higher frequency master clock than the bit clock, used to generate a more stable internal clock signal. The frequency of MCLK is a multiple of SCK, often used in DACs and ADCs.

In a typical I2S connection, at least three signal lines are required: SCK, WS, and SD.

2. Basic Working Principle of I2S I2S uses a synchronous communication mode, where data transmission is controlled by clock signals. The process follows these steps:

- **SCK clock drives data transmission:** The serial clock SCK determines the transmission time of each bit of data. During each SCK cycle, the I2S device transmits or receives one bit of audio data.
- **WS determines left and right channels:** The WS signal distinguishes between left and right channel data. When WS is low, left channel data is transmitted; when WS is high, right channel data is transmitted. Each frame of audio data contains both left and right channel data.
- **SD transmits data:** Audio data is transmitted bit by bit through the SD (Serial Data) line, typically MSB first.[12]

3. I2S Clock and Data Rate Relationship The clock in I2S is closely related to the audio data transmission rate, especially the sampling rate, bit depth, and number of channels, which determine the clock frequency settings.

- **Sampling Rate:** Indicates the number of samples per second, typically 44.1 kHz, 48 kHz, 96 kHz, or 192 kHz.
- **Bit Depth:** Indicates the number of bits per sample, such as 16-bit, 24-bit, or 32-bit.
- **Channels:** Indicates the number of audio channels, typically 2 channels (stereo) or more (e.g., 5.1 surround sound).

4. I2S Signal Timing The timing of I2S signals determines the synchronous transmission of data. The following diagram illustrates the typical I2S signal timing:

```
| SCK  | __|‾‾|__|‾‾|__|‾‾|__|‾‾|__|‾‾|__|‾‾|__|‾‾|__|‾‾|
| WS   | _____‾‾‾‾‾‾___________________________‾‾‾‾‾‾_____|
| SD   | _____Ldata(left channel)_____________Rdata(right channel)______||
```

- **SCK:** The clock signal controls the transmission of each bit of data.

- **WS:** The left/right channel selection signal for each frame of audio data. When WS is low, it indicates left channel data; when WS is high, it indicates right channel data.
- **SD:** The serial data line, transmitting left and right channel data under clock control.

5. I2S Communication Modes I2S supports multiple communication modes, mainly including:

- **Master Mode:** The master device generates all clock signals (including SCK, WS, and MCLK). For example, when the microcontroller acts as the master, it is responsible for generating clock signals and frame sync signals and transmitting them to the slave device.
- **Slave Mode:** The slave device receives the clock signals generated by the master and transmits data according to the clock signals. Audio codecs often work as slave devices.

6. I2S Data Formats I2S supports various data formats, commonly including:

- **16-bit, or 32-bit Data Formats:** Each transmission sends a sample, which may be 16-bit, or 32-bit data. The 16-bit format is the most common in audio applications.[11]
- **Left Justified Format:** The most significant bit (MSB) of the audio data is aligned with the edge of the frame sync signal.
- **Right Justified Format:** The least significant bit (LSB) of the audio data is aligned with the edge of the frame sync signal.

In this thesis, since mono audio was used, the sampling rate is 48 kHz and the bit depth is 16 bits.

5.3.4 DMA

DMA (Direct Memory Access) is a technology that allows devices to read from or write to memory directly, without CPU (Central Processing Unit) intervention[13]. DMA is very important in embedded systems as it can significantly improve data transfer efficiency, especially when transferring large amounts of data, such as audio, video, and sensor data. With DMA, the CPU can be freed up to handle other tasks instead of wasting time on data transfers.

1. Working Principle of DMA The core task of the DMA controller is to read data from the source address and write it to the destination address. During this process, the CPU only needs to initiate the data transfer request once, and the DMA controller will handle the entire data transfer process. Once the data transfer is complete, the DMA controller notifies the CPU through an interrupt.[13]

The basic steps of DMA operation are as follows:

1. **Initiate DMA Transfer:** The CPU configures the DMA controller, specifying the source address, destination address, the amount of data to transfer, the direction of data transfer, etc.[13]
2. **DMA Controller Executes the Transfer:** The DMA controller reads data from the specified source address and transfers it to the destination address.
3. **Data Transfer Completion:** When the DMA controller completes the data transfer, it triggers an interrupt to notify the CPU that the transfer is done. The CPU can then proceed with further processing.

Since DMA does not require CPU intervention, the entire process is much faster than manually moving data with the CPU.

2. Advantages of DMA The main advantages of using DMA for data transfer include:

- **Improved Efficiency:** DMA can perform data transfers in the background without consuming CPU time, allowing the CPU to continue executing other tasks.
- **Power Saving:** Since the CPU does not need to participate in each data transfer operation, the system can reduce CPU power consumption during DMA transfers.
- **Efficient Handling of Large Data Volumes:** DMA is particularly suitable for handling large-scale data transfers, such as audio or video streams. In such cases, using DMA can significantly increase transfer speed and efficiency.[14]

5.3.5 DMA Callback

DMA Callback is a mechanism used to notify the CPU of data transfer progress during DMA transmission. When using DMA to transfer data, callbacks can be used to handle the following situations:

- **Transfer complete:** When the data transfer is complete, the callback function notifies the CPU that the data can be further processed.

- **Half-transfer complete:** In some scenarios (such as audio processing or circular mode), when half of the transfer is completed, the callback is triggered to process the already transferred data.

- **Transfer error:** If an error occurs during the DMA transfer, the callback function can also be used to notify the CPU to handle the error.

5.3.6 Audio Process

PDM (Pulse Density Modulation) and PCM (Pulse Code Modulation) are two commonly used data formats in audio signal processing, especially in digital audio transmission and processing applications. PDM is often used in microphones and other audio sensors, while PCM is a common audio data format widely used for playback, storage, and processing of digital audio. The conversion from PDM to PCM is a common task in many embedded systems (such as STM32), especially when using digital microphones. The following sections provide a detailed explanation of PDM, PCM, and their conversion.

1. PDM (Pulse Density Modulation)

Concept of PDM PDM is a modulation technique used for audio signals, which uses a high-frequency pulse stream to represent the density of the audio signal. The core idea of PDM is to use the density of the pulses to express the intensity of the audio signal: In a PDM signal, each bit is a pulse:

- 1 represents a higher signal strength (corresponding to a higher level in the analog signal).

- 0 represents a lower signal strength (corresponding to a lower level in the analog signal).

At high frequencies, the PDM pulse stream can approximate the original analog audio signal. The higher the density of the pulse stream, the higher the audio signal level it represents; conversely, the lower the density of the pulse stream, the lower the audio signal level it represents.

Advantages of PDM

- **Simple hardware implementation:** PDM is a very simple modulation method, making it particularly suitable for use in microphones and other audio sensors. Digital microphones using PDM technology only need to output a binary signal (1 or 0), making the hardware implementation straightforward.

- **High bandwidth:** The sampling rate of PDM is much higher than the target audio rate, allowing the original audio signal to be recovered through digital filtering.

Disadvantages of PDM

- **Requires post-processing:** Although generating PDM signals is simple in hardware, more complex digital signal processing is required afterward, especially the conversion from PDM to PCM format, which requires substantial computational resources.
- **High noise:** PDM signals are essentially pulse signals, which can introduce significant noise if directly stored and processed. Typically, filters are needed to reduce noise.

2. PCM (Pulse Code Modulation)

5.3.7 Concept of PCM

PCM is a more common audio data format, where the analog audio signal is sampled and quantized to generate a digital representation. Each sample point in PCM is typically a multi-bit binary value that accurately represents the audio signal level.

PCM adopts discrete sampling and discrete quantization:

- **Discrete sampling:** A certain number of audio samples are collected per second, known as the sampling rate (e.g., 44.1 kHz, 48 kHz).
- **Discrete quantization:** The audio signal level at each sample point is mapped to a fixed bit depth (e.g., 16-bit, 24-bit, or 32-bit).

Advantages of PCM

- **Widely used:** PCM is the most common audio format, supported by almost all audio playback devices and audio processing software.
- **High audio quality:** With an appropriate sampling rate and bit depth, PCM can accurately represent the audio signal with minimal quality loss.

Disadvantages of PCM

- **Large storage space:** Compared to PDM, PCM requires more data storage. Each sample point is represented by multiple bits, consuming more storage space and transmission bandwidth.

3. Comparison between PDM and PCM

The table of the comparison between PDM and PCM is shown below.

Characteristic	PDM (Pulse Density Modulation)	PCM (Pulse Code Modulation)
Signal Type	1-bit pulse density signal (high frequency)	Multi-bit sampled values (typically 16-bit or 24-bit)
Generation Method	High-frequency pulse stream, density represents signal strength	Sampling and quantization of audio signals
Bandwidth	Lower data volume but requires high-frequency filtering and demodulation	Consumes more bandwidth and storage space
Audio Quality	Needs demodulation, original signal has higher noise	Accurately represents audio signals, high quality
Typical Applications	Digital microphones, audio sensors	Audio storage, playback, transmission

Using PDM is for the transmission of audio data. After saving as PDM, it will be stored in the output buffer and transferred via USB.

5.3.8 USB Configuration

1. USB Basic Architecture

USB is a host-controlled bus architecture where the host (typically a PC or embedded host) communicates with peripheral devices (such as a mouse, keyboard, audio devices, etc.). USB communication is comprised of two main parts:

- **Host Side (Host):** Responsible for managing the bus, polling devices, managing data transfers, and allocating bandwidth.
- **Device Side (Device):** External devices recognized and communicated with by the host, such as microphones, storage devices, cameras, etc.

2. USB Configuration

USB configuration is the way a device reports its capabilities and functions to the host after being detected. USB devices inform the host of their functionality and required resources through a series of descriptors. These descriptors are structured data that the host reads, and based on this information, the host allocates an

address and bandwidth for the device.
In this thesis, USB is configured to enable efficient transmission between STM32 and PYNQ.
The I2S interface and DMA are configured for data transfer. Audio data is transmitted to the memory buffer via I2S and DMA. When the DMA transfer is halfway complete, it triggers the process of converting PDM data to PCM and stores it in the buffer. Afterward, the content is stored in the output buffer and then sent to the USB.

5.4 System Process After Obtaining Audio

On the PYNQ-Z2, the `FLAG` is initialized to 0 and the system continuously monitors the audio input device. When `FLAG=0` and `button3` is pressed, it indicates the task is complete, and the program will exit. If not, the program will continue reading the data coming from the audio device. Next, it distinguishes between noise and human voice. Since under normal conditions, the decibel levels of human voice and noise are different, with background noise being much lower than actual human voice, a condition is set: if the signal strength is greater than 700, greater than the `max_signal`, and this occurs 5 consecutive times, it is considered human voice. At this point, the audio data is saved, and recording stops.Save the data as a WAV file.
Since the recorded audio has a relatively high volume, in speech processing, higher volume means greater interference. Therefore, the original speech needs to be appropriately processed. The RMS energy should be reduced and saved to a new file.Then encode it into spike pulse signals.And save it as a new json file.Finally,run the model file, read the generated spike data, and output the recognition number.The real time recognization work flow is shown below.

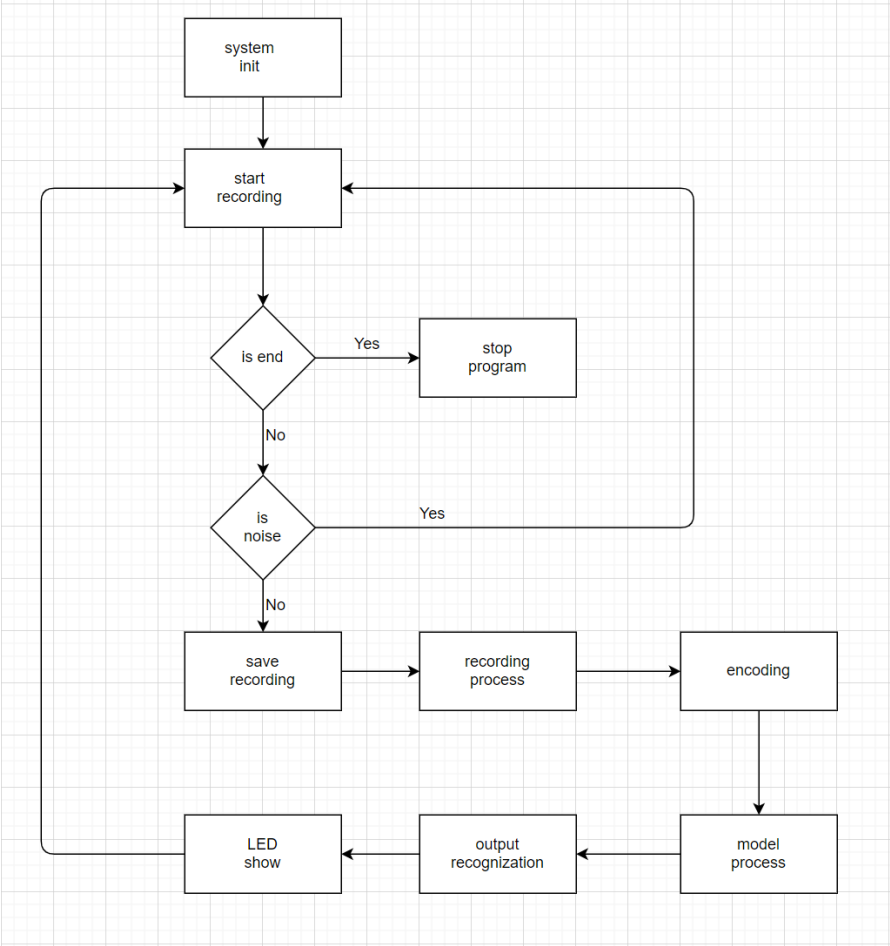

Figure 5.2: real time recognization

5.4.1 System Detail

The system used by the PYNQ-Z2 is v3.0.1, which is a Linux-based operating system derived from Ubuntu. Additionally, PYNQ comes with built-in libraries that can be used directly to control hardware. For example, in this thesis, after recognition is completed, the output results will be displayed using LEDs and other indicators.
In PYNQ, in order to use a USB microphone, it is necessary to install the `alsa-lib`, `pyaudio`, and `wave` packages.
When using the model, due to the fact that PyTorch is not supported under Python version 3.10 on the current system, Python version 3.7 is used to install PyTorch.
During system initialization, the audio data format is set to `pyaudio.paInt16`, with the channel set to mono, a sampling rate of 48,000 Hz, and a cache size of

512KB.And the recording device is detected.

5.4.2 Start Recording

The LED lights play an important signaling role in this system. After the system starts, the LEDs will turn off sequentially, and then turn on in sequence, indicating that the system has completed the startup process and has entered the recording program.

5.4.3 Is End Program

Since the system automatically recognizes speech in real-time and outputs predicted values, it needs a way to stop its operation. Therefore, before starting to check the recording, a button is used to determine whether there is a signal to stop the entire task. If the button is pressed, it indicates that the speech recognition task should be terminated. If not, the system continues to check the audio input.

5.4.4 Is Noise

In the process of real-time system processing, distinguishing between noise and speech is a crucial part. This not only determines the final accuracy of the system but also its overall usability. Moreover, detecting the endpoint of speech is equally important, as it directly affects the accuracy of the model's subsequent judgments. The main flowchart for handling noise is shown below.

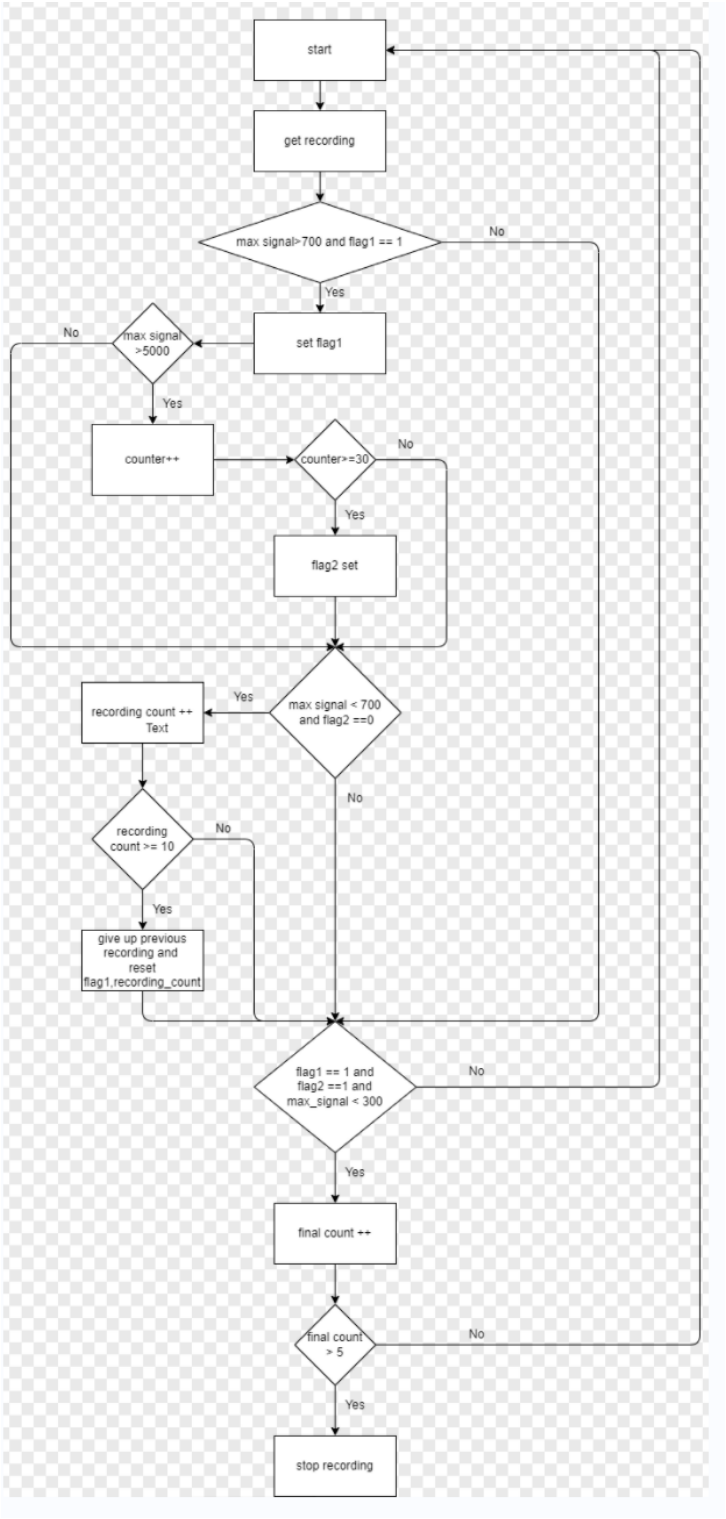

Figure 5.3: Noise Check

In this system, the determination of noise is based on signal strength and continuity. Through observation, it was found that under normal conditions, the strength of noise is typically below 700. Therefore, if the signal strength is below 700, the current recording is not saved. When the strength exceeds 700, it indicates the start of a sound, which may be human voice or noise, and is uncertain at this point. However, data needs to start being saved. At this time, `flag1` is set, indicating that data saving has started.
Once data saving begins, if the sound is a normal human voice, the max signal strength will exceed 5000. However, if the strength drops below 700 and remains there for 10 consecutive instances, it is considered transient noise, and the previously saved data should be discarded, resetting all flags.
After the sound strength exceeds 5000 for more than 30 consecutive instances, `flag2` is set, indicating that recording has been successfully captured and is now waiting to stop. At the end of the sound, the maximum strength will drop below 300, and if this happens for 5 consecutive instances, it indicates that the recording has ended.

5.4.5 Save Recording And Recording Process

After saving the audio, due to the high volume of the original sound, noise has a significant impact in this situation. Therefore, it is necessary to reduce the volume. The method is to obtain the current audio's RMS value, with the formula:

$$\text{final\_signal} = \text{signal} \times \left(\frac{2.3}{\text{current\_rms}}\right)$$

The value 2.3 was obtained through experimentation and comparison with the training data. After processing, the RMS strength of the data will be close to that of the model's training set.

5.4.6 Encoding And Model Process

After getting the audio, the system using the spike encoding based on MFCC to encode the audio.Then saving the data as JSON file, the model program is launched using subprocess, and the result is returned.

5.4.7 LED Show

After the model returns the recognition number, the program will display it in binary using the LEDs. Initially, all the LEDs are on, so this starting state does not affect the recognition task. For example, 1 is represented as `0001`, with only LED0 on and the others off. 9 is represented as `1001`, meaning LED0 and LED3

are on while the others are off. For 0, all LEDs are off.
After recognition, the program re-enters the state of waiting for audio input to perform the next recognition, continuing this process until the task termination button is detected as pressed.

Chapter 6

Fine-tune And Discussion And Analysis Of Research Results.

Due to the differences in recording devices and recording environments between the previous training and testing datasets and the current system, the recognition performance was not satisfactory. To address this issue, the author recorded a new dataset using the current system and fine-tuned the model with this recorded data, achieving higher accuracy. The specific recordings are categorized into three types:

1. The author recorded a total of 1,000 samples, with 100 samples for each digit. Among these, 80% were used as the fine-tuning training set, and 20% were used as the fine-tuning test set.

2. The author's wife recorded a total of 1,000 samples, with 100 samples for each digit. Among these, 80% were used as the fine-tuning training set, and 20% were used as the fine-tuning test set.

3. A combined dataset of 2,000 samples recorded by both the author and his wife, with 200 samples for each digit. Among these, 80% were used as the fine-tuning training set, and 20% were used as the fine-tuning test set.

The fine-tuning workflow are as follows:

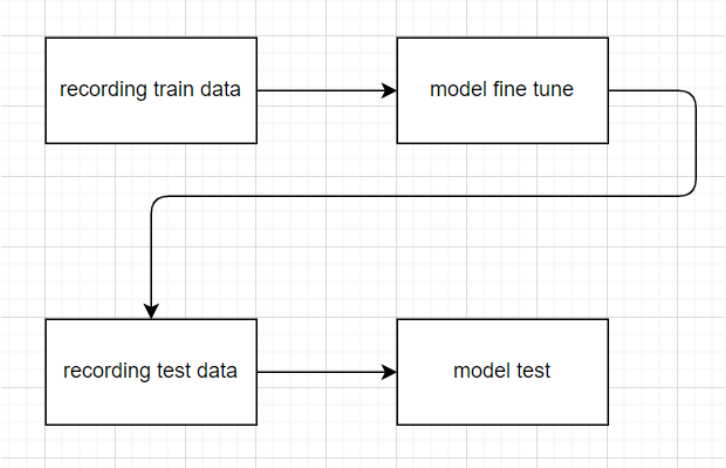

Figure 6.1: fine tune

After fine-tuning, the obtained data are as follows:

Recording Type	Accuracy
One person's recording (female)	94%
One person's recording (male)	93%
Two people's recording (one male, one female)	93%

Table 6.1: Accuracy of recordings by gender and group

Current Data Explanation

1. The accuracy after fine-tuning slightly decreases, which may be related to the small amount of data.

2. The results of fine-tuning with recordings from two people and from one person (regardless of gender) are essentially consistent, indicating that the system is relatively stable and does not exhibit significant fluctuations due to fine-tuning with different datasets.

Chapter 7

Conclusions

The speech recognition system implemented based on the methods in this thesis has been verified to function correctly and recognize speech accurately. During testing, although occasional errors occurred, the overall recognition accuracy remained high.

As for the model, the SNN models used in this thesis are all pre-existing models, and no adjustments were made to them. SNN is still in the research stage. Some say that SNN is the third-generation neural network model, and I firmly believe this is the right direction. As we know, current ANNs, while achieving exciting results in areas like semantic recognition and multimodality, also consume a significant amount of energy, which cannot be ignored. In contrast, SNNs have much lower power consumption than ANNs. From the perspective of human evolution, our brain must have evolved towards maximum efficiency. And SNNs are more similar to the human brain in their operation than ANNs, although they still require further research.

For example, models like LIF use formulas that are overly simplified, limiting improvements in accuracy. On the other hand, the adLIF model, although it uses an adaptive variable, still operates within a fixed range for its random variables. Whether this range is scientifically valid is debatable, or more precisely, how the variables should change within this range at different times needs a better expression to describe it. In RLIF, although the model introduces connections between neurons, it still has an alpha, i.e., the decay rate of the membrane potential, which is a random number. These random numbers in human brain cells are certainly not random, so exploring the correct way to express these random numbers could be an important direction. If these issues can be resolved, I believe SNNs will see even more exciting developments.

Regarding data processing, this thesis proposes a new processing method based on the original data encoding, which significantly improves the model accuracy.

Although this method has achieved good results in speech recognition, from the

perspective of overall SNN development, in comparison to ANN, it is necessary to explore the fields of NLP and image recognition, and then expand to multimodal tasks. Therefore, how to unify the encoding of speech, semantics, and images will be an interesting direction.
In terms of hardware, the overall architecture provided a reference for future use. Currently, since the training dataset is mono-channel, both training, testing, and system operation have been conducted in mono-channel. If multi-channel training data becomes available in the future, the hardware can also be adapted to support multi-channel usage.

Bibliography

[1] Dario Padovano, Alessio Carpegna, Alessandro Savino, and Stefano Di Carlo. «SpikeExplorer: Hardware-Oriented Design Space Exploration for Spiking Neural Networks on FPGA». en. In: *Electronics* 13.9 (Jan. 2024). Number: 9 Publisher: Multidisciplinary Digital Publishing Institute, p. 1744. ISSN: 2079-9292. DOI: 10.3390/electronics13091744. URL: https://www.mdpi.com/2079-9292/13/9/1744 (visited on 09/06/2024) (cit. on p. 4).

[2] Alexandre Bittar and Philip N Garner. «A surrogate gradient spiking baseline for speech command recognition». In: *Frontiers in Neuroscience* 16 (2022). DOI: 10.3389/fnins.2022.865897 (cit. on pp. 4, 6, 7, 21, 23, 25, 26).

[3] PYNQ. *PYNQ Boards*. https://www.pynq.io/boards.html. Accessed: 2024-10-12. 2024 (cit. on pp. 9, 10).

[4] B. Cramer, Y. Stradmann, J. Schemmel, and F. Zenke. «The Heidelberg Spiking Data Sets for the Systematic Evaluation of Spiking Neural Networks». In: *IEEE Transactions on Neural Networks and Learning Systems* (2020), pp. 1–14. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2020.3044364 (cit. on pp. 11, 13, 14, 17).

[5] Zrar Kh. Abdul and Abdulbasit K. Al-Talabani. «Mel Frequency Cepstral Coefficient and its Applications: A Review». In: *IEEE Access* 10 (2022), pp. 122136–122158. DOI: 10.1109/ACCESS.2022.3223444 (cit. on p. 12).

[6] Qinan Wang. «Application of Advanced Material in Flexible Fully Solution Processed Thin Film Transistor». PhD thesis. The University of Liverpool (United Kingdom), 2023 (cit. on p. 13).

[7] Benjamin Cramer, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke. «The Heidelberg Spiking Data Sets for the Systematic Evaluation of Spiking Neural Networks». In: *IEEE Transactions on Neural Networks and Learning Systems* 31.9 (2020), pp. 3292–3301. DOI: 10.1109/TNNLS.2020.2966017 (cit. on pp. 15, 29).

[8] Liqun Luo. *Principles of Neurobiology*. 2nd. CRC Press, 2020 (cit. on p. 15).

[9] Shashi Chunduri. *Understanding LSTM: Plain and Simple*. https://medium.com/@chunduri11/understanding-lstm-plain-and-simple-96026b4468c6. Accessed: 2024-10-12. 2024 (cit. on p. 19).

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. «Attention Is All You Need». In: *arXiv preprint arXiv:1706.03762* (2017). URL: https://arxiv.org/abs/1706.03762 (cit. on p. 19).

[11] STMicroelectronics. *STM32F446RE*. https://www.st.com/en/microcontrollers-microprocessors/stm32f446re.html. Accessed: 2024-10-12. 2024 (cit. on pp. 34, 41).

[12] Jianwei Zhang, Yi Yi, and Yujie Zhi. «Improved Design of I2S IP with Sidetone Signal Processing Capability (I2S_SC) for ASICs». In: *2022 IEEE 8th International Conference on Computer and Communications (ICCC)*. IEEE, 2022, pp. 1391–1395. DOI: 10.1109/ICCC55550.2022.10026738 (cit. on pp. 39, 40).

[13] Patricio Bulić. *Understanding Computer Organization*. Springer Science and Business Media LLC, 2024 (cit. on pp. 41, 42).

[14] D.A. Bradley, N.C. Burd, D. Dawson, and A.J. Loader. *Mechatronics - Electronics in Products and Processes*. 2nd. CRC Press, 2018 (cit. on p. 42).