|  |  |
| --- | --- |
| **DE-43 (EE) Abdul Wasay Naveed, Sudais Waseem, Zahid Gul, Wajahat Khan** | ­­­­­­­­­­­­­­­­­­  **IMPLEMENTATION OF POLAR CODES IN BATTERY CONSTRAINED DEVICES USING FNN**    **COLLEGE OF**  **ELECTRICAL AND MECHANICAL ENGINEERING NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY RAWALPINDI**  **2025** |

**COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING**



**DE-43 EE**

**PROJECT REPORT**

**IMPLEMENTATION OF POLAR CODES IN BATTERY CONSTRAINED DEVICES USING FNN**

Submitted to the Department of Electrical Engineering in partial fulfillment of the requirements for the degree of

**Bachelor of Engineering**

**in**

**Electrical**

**2025**

**Supervisor:**

Engr. Furqan Haider Qureshi

**Co-Supervisor:**Ma’am Sobia Hayee

**Members:**

Abdul Wasay Naveed

Sudais Waseem

Zahid Gul

Wajahat Khan

**CERTIFICATE OF APPROVAL**

It is to certify that the project **“IMPLEMENTATION OF POLAR CODES IN BATTERY CONSTRAINED DEVICES USING FNN”** was done by **Abdul Wasay Naveed,**

**Sudais Waseem, Zahid Gul, Wajahat Khan** under the supervision of **Engr. Furqan Haider Qureshi** and co-supervision of **Ma’am Sobia Hayee**

Submission: This project was submitted to the College of Electrical and Mechanical Engineering, National University of Sciences and Technology, Pakistan, as part of the requirement for the degree of Bachelor of Electrical Engineering.

**Students:**

## 1. Abdul Wasay Naveed

NUST ID: 366696 Signature:

## 2. Sudais Waseem

|  |  |
| --- | --- |
| **3. Zahid Gul** |  |
| NUST ID: 377141 | Signature: |

NUST ID: 369142 Signature:

|  |  |
| --- | --- |
| **4. Wajahat Khan** |  |
| NUST ID: 368651 | Signature: |

**Approved By:**

Project Supervisor: **Engr. Furqan Haider Qureshi**

Signature: Date:

Head of Department: **Dr. Qasim Umer Khan**

Signature: Date:

**DECLARATION**

We declare that no part of this project thesis has been submitted in support of an application for another degree or qualification. We have not submitted this thesis to any other university or educational institution. We are totally liable for any disciplinary action taken against us based on the nature of the proved offence, including the revocation of our degree.

**Students:**

## 1. Abdul Wasay Naveed

NUST ID: 366696 Signature:

## 2. Sudais Waseem

|  |  |
| --- | --- |
| **3. Zahid Gul** |  |
| NUST ID: 377141 | Signature: |

NUST ID: 369142 Signature:

|  |  |
| --- | --- |
| **4. Wajahat Khan** |  |
| NUST ID: 368651 | Signature: |

**ACKNOWLEDGMENTS**

We would like to extend our biggest thanks to all persons involved in the successful completion of our final year project; Implementation of Polar Codes in Battery Constrained Devices Using Feedforward Neural Networks. Many individuals and institutions were of invaluable support in the realization of this project.

Before all we would like to express our deep gratitude to our supervisor; Engr. Furqan Haider Qureshi, for an excellent mentorship, technical advice and continuous motivation during this work. He was very knowledgeable with some constructive feedback that helped us understand and push the boundaries of interfacing Polar Codes with neural networks in an embedded system.

We are equally thankful to our co supervisor, **Ma’am Sobia Hayee**, for guiding us through the whole project with such a knowledgeable way. Her clarity of thought and great support helped us out the difficult periods and pushed our work more precisely.

.

Firstly, we would like to acknowledge the **Department of Electrical Engineering** for providing us with physical resources, infrastructure and nurturing environment for doing research and innovation. The project was done effectively due to the department’s commitment to academic excellence.

We also wish to thank our dedicated team: **Zahid Gul, Wajahat Khan, Abdul Wasay Naveed** and **Sudais Waseem**. Zahid and Sudais laid a strong theoretical basis of Polar Codes, Wajahat and Sudais worked on the development of efficient decoding algorithms, Abdul Wasay was responsible for preparing the datasets and training of the Feedforward Neural Network under the supervision of zahid who worked towards the hardware implementation on ESP32 with LoRa SX1278 using the Arduino IDE to realise our vision and make it a functional prototype. The process of adopting this project involved key collaboration, commitment and mutual respect among my team.

Moreover, we are thankful for our classmates and friends who supported us morally, provided constructive feedback and cooperated with us during the brainstorming sessions and testing times. Their presence was extremely handy for long coding marathons and system debug.

Finally, we wish to extend our dedicated appreciations to our families always ready to share their love, unfailing patience and unwavering encouragements throughout the demanding process.

We are also thankful to open source community and to authors of different research papers, tools and libraries, such as **NumPy, Pandas, Scikit-learn, TensorFlow, Matplotlib**. Collective contributions of all members gave us a foundation and work floors we could easily use in exploring, learning and innovating.

Finally, this project has been marked by collaboration, interdisciplinary learning, support from an inspiring academic and personal network. This is a journey for which we should be very lucky to have lived.

## ****ABSTRACT****

Driven by the rapid growth of the Internet of Things (IoT), along with the arrival of 5G communication, it is extremely important to ensure reliable and energy efficient data transmission in energy shortage embedded devices. We will investigate how to implement Polar Codes, a class of capacity achieving error correcting codes, in ultra-low power wireless systems in this project. Instead of fixed decoding algorithm selection, we propose an intelligent decoding selection driven by a Feedforward Neural Network (FNN) that dynamically makes decoding algorithm choice according to the real time communication feedback, in order to reach the tradeoff between decoding accuracy and power consumption.

Amongst those decoding algorithms examined in this study are Successive Cancellation (SC), Successive Cancellation List (SCL) and List Viterbi (LV) with their respective performance and computational complexity tradeoffs. It is found that LVA and SCL perform better in terms of bit error rate (BER) but they are power hungry and not suitable for continuous deployment in resource constrained environments. While SC is much less power intensive, it suffers from severe performance degradation in noisy environment. Our training of this ulterior FNN model predicts the optimal decoder for any transmission instance based on packet retries, packet loss, ACK success, and CRC error rate.

BPSK modulation and a realistic wireless channel with Rayleigh fading along with Additive White Gaussian Noise (AWGN) is used to simulate the communication system in Python. In this case, the Bhattacharyya parameter is used to choose the frozen bits in the Polar encoder. The decoders are evaluated individually and compared with that of FNN selected decoder in terms of BER, outage probability, power delay profile (PDP) and average power consumption. Results are averaged over multiple trials to ensure statistical relevance and the study is performed within an SNR range of 0 to 30 dB, for which extensive simulations are performed.

The model is integrated into a hardware prototype of an ESP32 microcontroller with LoRa SX1278 transceiver suitable with low power communication over long distance. Realistic energy models for the embedded devices are used in the profiling of the power consumption, based on both computational load and radio usage.

The FNN based decoder selection gives a BER almost as good as LVA and SCL under high noise conditions and its power footprint is comparable with SC under good channel conditions. It shows this energy‐efficiency/accuracy balance can be achieved by applying machine learning to coding theory and is thus a proof of concept for adaptive wireless communication enabled by machine learning and coding theory in power constrained battery systems.

**SUSTAINABLE DEVELOPMENT GOALS**

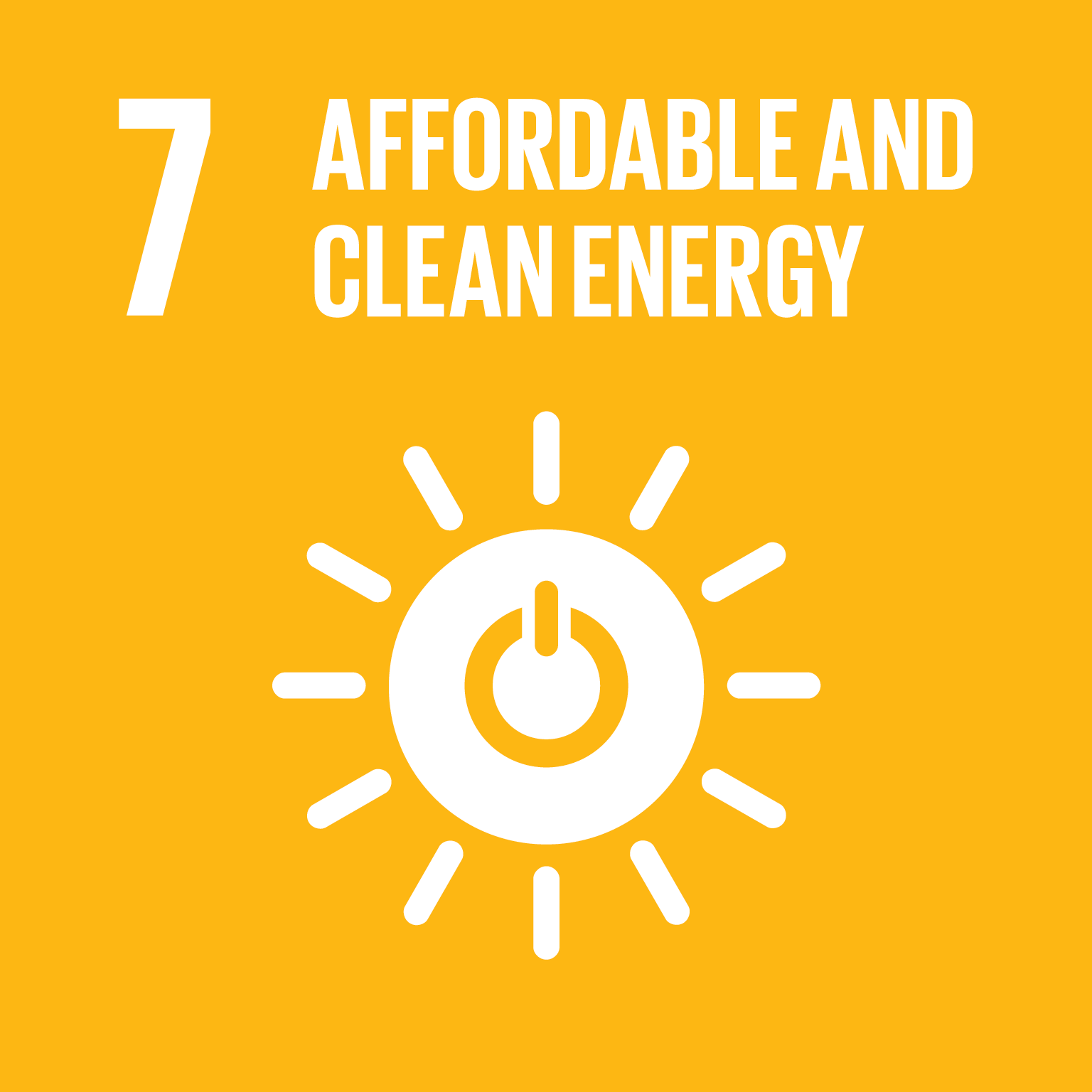
**SDG 9 – Industry, Innovation, and Infrastructure:**

By implementing advanced Polar Code decoding techniques on low-cost embedded hardware using machine learning, our project promotes innovation in communication systems and contributes to building resilient, intelligent infrastructure for IoT and industrial automation.



**SDG 7 – Affordable and Clean Energy:**

The focus on minimizing power consumption through intelligent decoder selection helps extend battery life in embedded devices, supporting the development of energy-efficient technologies crucial for clean and affordable energy use in wireless systems.



**SDG 11 – Sustainable Cities and Communities:**

With its low-power, long-range communication capabilities using ESP32 and LoRa SX1278, our system can be deployed in smart city applications such as environmental monitoring and public infrastructure, enhancing sustainability and reducing maintenance needs.



**TABLE OF CONTENTS**

## DECLARATION------------------------------------------------------------------------------------------------ iv

## ACKNOWLEDGMENTS ------------------------------------------------------------------------------------- v

## ABSTRACT------------------------------------------------------------------------------------------------------ vi

## SUSTAINABLE DEVELOPMENT GOALS-------------------------------------------------------------- vii

## TABLE OF CONTENTS-------------------------------------------------------------------------------------- viii

## ****Chapter 1: Introduction****------------------------------------------------------------------------------------ 1

1.1 Background ------------------------------------------------------------------------------------------------ 1   
1.2 Motivation---------------------------------------------------------------------------------------------------2  
1.3 Problem Statement---------------------------------------------------------------------------------------- 2  
1.4 Objectives--------------------------------------------------------------------------------------------------- 4  
1.5 Scope of Work--------------------------------------------------------------------------------------------- 4  
1.6 Research Questions--------------------------------------------------------------------------------------- 6  
1.7 Methodology Overview--------------------------------------------------------------------------------- 7  
1.8 Thesis Organization-------------------------------------------------------------------------------------- 8

## ****Chapter 2: Literature Review****---------------------------------------------------------------------------- 9

2.1 Introduction to Channel Coding----------------------------------------------------------------------- 9  
2.2 Evolution of Error Correction Codes----------------------------------------------------------------- 9  
2.3 Polar Codes and Their Importance in 5G---------------------------------------------------------- 10  
2.4 Review of Polar Decoding Techniques------------------------------------------------------------- 11  
2.5 Neural Networks in Communication Systems---------------------------------------------------- 16  
2.6 Energy Constraints in IoT Devices------------------------------------------------------------------ 21  
2.7 Existing Works on Decoder Selection-------------------------------------------------------------- 22  
2.8 Research Gap and Contribution---------------------------------------------------------------------- 22

## ****Chapter 3: System Architecture and Methodology****----------------------------------------------- 24

3.1 Overview of the Proposed System------------------------------------------------------------------- 24  
3.2 Communication Model--------------------------------------------------------------------------------- 25  
  3.2.1 Message Generation and Encoding  
  3.2.2 Modulation (BPSK)  
  3.2.3 Channel Model: Rayleigh + AWGN  
3.3 Decoder Implementations----------------------------------------------------------------------------- 29  
  3.3.1 SC Decoder  
  3.3.2 SCL Decoder  
  3.3.3 LV Decoder  
3.4 Feature Selection and Dataset Preparation-------------------------------------------------------- 31  
  3.4.1 Retries  
  3.4.2 Packet Loss  
  3.4.3 ACK Success  
  3.4.4 CRC Error  
3.5 Frozen Bit Selection Using Bhattacharyya Parameters---------------------------------------- 33  
3.6 Neural Network Design------------------------------------------------------------------------------- 34  
  3.6.1 FNN Architecture  
  3.6.2 Input Normalization  
  3.6.3 Output Labels (Decoders)  
  3.6.4 Training Methodology  
3.7 Decoder Prediction Logic----------------------------------------------------------------------------- 36  
3.8 Hardware Integration (ESP32 + LoRa SX1278) ------------------------------------------------ 37  
3.9 Power Modeling and Energy Profiling------------------------------------------------------------- 38

## ****Chapter 4: Simulation Environment****------------------------------------------------------------------ 40

4.1 Tools and Libraries Used------------------------------------------------------------------------------ 40  
  4.1.1 Python  
  4.1.2 TensorFlow  
  4.1.3 Pandas and NumPy  
  4.1.4 Scikit-learn  
  4.1.5 Matplotlib  
  4.1.6 Arduino IDE  
4.2 Simulation Parameters---------------------------------------------------------------------------------- 41  
4.3 Flow of Simulation-------------------------------------------------------------------------------------- 41  
4.4 Evaluation Metrics-------------------------------------------------------------------------------------- 44  
  4.4.1 Bit Error Rate (BER)  
  4.4.2 Outage Probability  
  4.4.3 Power Delay Profile (PDP)  
  4.4.4 Power Consumption

## ****Chapter 5: Results and Analysis****------------------------------------------------------------------------46

5.1 BER Comparison---------------------------------------------------------------------------------------- 46  
  5.1.1 SC vs. SCL vs. LVA  
  5.1.2 FNN-Predicted Decoder vs. Others  
5.2 Outage Probability Across SNR--------------------------------------------------------------------- 48  
5.3 Decoder Selection Trends by FNN------------------------------------------------------------------ 49  
5.4 Power Consumption vs. SNR------------------------------------------------------------------------- 50  
  5.4.1 Individual Decoders  
  5.4.2 FNN-Driven Selection  
5.5 Power Delay Profile Analysis------------------------------------------------------------------------ 52  
5.6 Statistical Interpretation-------------------------------------------------------------------------------- 53  
5.7 Visualizations--------------------------------------------------------------------------------------------- 54  
  5.7.1 BER Graphs  
  5.7.2 Outage Graphs  
  5.7.3 Power Graphs  
  5.7.4 PDP Plots  
5.8 Discussion of Results----------------------------------------------------------------------------------- 55

## ****Chapter 6: Hardware Implementation****--------------------------------------------------------------- 56

6.1 ESP32 Microcontroller--------------------------------------------------------------------------------- 56  
6.2 LoRa SX1278 Communication Module------------------------------------------------------------ 57  
6.3 Hardware Power Profiling----------------------------------------------------------------------------- 57  
6.4 Integration of Neural Network Logic--------------------------------------------------------------- 58  
6.5 Deployment Constraints and Solutions------------------------------------------------------------- 59  
6.6 Prototype Testing and Observations---------------------------------------------------------------- 59

## ****Chapter 7: Conclusion and Future Work****------------------------------------------------------------60

7.1 Summary of Work--------------------------------------------------------------------------------------- 60  
7.2 Key Contributions--------------------------------------------------------------------------------------- 60  
7.3 Limitations------------------------------------------------------------------------------------------------ 61  
7.4 Future Improvements----------------------------------------------------------------------------------- 62  
  7.4.1 Use of Real-Time SNR Estimation  
  7.4.2 Online Learning or Reinforcement Learning  
  7.4.3 Extension to LDPC and Turbo Codes  
  7.4.4 Hardware Optimization for Ultra-Low Power

## ****References****

(All research papers, books, online documentation, and tools used)-----------------------------**64**

## ****Appendices****-----------------------------------------------------------------------------------------------------65

**Appendix A:** FNN Model Architecture---------------------------------------------------------------- **65**  
**Appendix B:** Sample Code Snippets-------------------------------------------------------------------- **66**

**CHAPTER 1: INTRODUCTION**

* 1. **Background**

**1.1.1 The Role of Forward Error Correction in Communication Systems**

Since past data integrity in digit motions is acquainted with a fuzzy example, Forward Error Correction (FEC) strategies are fundamental for most logical passageways for the most part over a harsh channel like radio or satellite spaces. This redundancy is structured to the data being transmitted so that receivers can not only detect errors but also correct them without feedback or retransmission, which is important for real time or high latency systems.

Polar Codes (introduced by Erdal Arıkan in 2009) have recently emerged out as one of the FEC Codes (such as Turbo, LDPC, Reed-Solomon) enjoying a great level of popularity. The first class of codes proven to achieve the Shannon capacity over a wide subclass of channels under a low complexity of the decoding algorithm Successive Cancellation (SC).

Polar Codes differ in their novel channel polarization principle, which takes a physical channel and combines it and splits it into good and bad synthetic channels. The information bits pass through the good channels, but the bad channels are "frozen" with predetermined values; generally, zeros. Polar Codes leverage this structured use of frozen bits to gain large margin in speed, as shown in binary input symmetric memoryless channels with high gain in 5G and beyond.

*1.1.2 Polar Codes in 5G and IoT*

Due to their capacity achieving properties and their efficient decoding algorithms (Successive Cancellation, SCL, SC Flip, etc.) Polar Codes were chosen for coding of the control channel (Control Chanel Coding, CCC) in the 5G New Radio (NR) standard.

However, their use in the low power IoT devices is challenging. Although Polar Codes achieve excellent error correction performance, SCL and LVA algorithms used for their decoding can incur higher complexity and memory requirements that are energy expensive and increase processing delay in constraint environments.

This introduces a trade-off:

* LVA based decoders are high performing, but power hungry.
* There are low complexity decoders (e.g., SC) that save power while suboptimal in noisy channel.

Therefore, it is very important to implement intelligent decoder selection mechanisms, adapting the decoding to channel conditions and device constraints according to the energy and reliability requirements at hand. Practical implementations, and especially IoT or battery operated systems, need such intelligent decoder selection mechanisms.

*1.1.3 Battery Constraints in Embedded Devices*

As a low cost, ultra-low power and long range wireless communication device, ESP32 with LoRa SX1278 transceiver is commonly used in modern IoT network due to their uses.

Deployment of these devices is often to remote or hard to get to environments such as agriculture, smart metering and environmental monitoring where power source is limited. Consequently, they rely on small batteries or energy harvesting sources and are expected to be operational autonomously for months or in some cases years without human intervention.

With this, power efficiency becomes an important design factor for all communication subsystems, which in turn are one of the most energy consuming components. The battery life can be significantly drained if not run very intelligently by running these complex decoding algorithms (like the LVA or the SCL).

That means, adaptive, energy aware decoding tactics need to be implemented that will trade off between the error performance and power constraints, while at the same time transmitting reliable information***.***

* 1. **Motivation**

*1.2.1 Energy-Efficient Reliable Communication*

However, in current communication system, there is a tradeoff between energy efficiency and decoding performance, and especially at the resource constrained IoT devices.

On the other hand, high performing decoders like List Viterbi Algorithm (LVA) and Successive Cancellation List (SCL) are able to correct a large number of bits with small error but require heavily computational resource and power, which is not applicable for battery powered devices.

At the same time, error correction performance of lightweight decoders such as Successive Cancellation (SC) degrades quickly under noisy or unpredictable channel conditions while these decoders are energy efficient and easy to implement.

To overcome this gap, this project uses a machine learning based solution that selects the most suitable decoder at runtime using the real time channel features to trade power consumption with communication reliability. The system to be proposed will take advantage of the Feedforward Neural Network to predict the best decoder avoiding the inefficiency on energy while guaranteeing robustness in the environment.

*1.2.2 Intelligence at the Edge*

This system is made capable of real time prediction of optimal decoder based on current link conditions (e.g., SNR, retries, packet loss) via addition of a Feedforward Neural Network (FNN).

This allows the system to:

* Autonomously adapt to a changing channel environment.
* Select low-power decoders like SC during stable transmission

LVA or SCL robust decoders can be used in place of them as error probability begins to increase.

Consequently, the device balances between performance and efficiency in real time without sacrificing on the reliability of communication, hence optimizing energy usage.

**1.3 Problem Statement**

Most classes of modern embedded and IoT devices (such as ESP32 + LoRa) run on tight energy budgets. However, high performance decoders (as SCL or List Viterbi) are generally demanded for reliable communication and especially under fading and noisy conditions. The problem with these decoders, however, lies in the fact that they are too computationally intensive and power hungry for such devices.

The other end are the lightweight low power decoders such as SC, which fail over the harsh channel. Thus, considerable tradeoff exists between energy and accuracy.

### **Project Goal:**

**Development of an intelligent system which select the Polar Code decoder (SC, SCL, LVA) at runtime adaptively using current link conditions — achieve optimal efficiency without sacrificing performance.**

**Sub-Problems to Address:**

**Performance under Realistic Channels:**

The multipath fading (Rayleigh) and the Gaussian noise (AWGN) influence wireless channels.

Under both these effects, the project investigates how SC, SCL, and LVA behave with respect to the level of SNR.

This specifies in practice when each decoder should be used.

**Can ML Predict Optimal Decoder?**

Feedforward Neural Network (FNN) is trained with the link layer metric (e.g. SNR, retries, packet loss).

In particular, the question is if it can accurately do this for the current channel conditions.

The key point to highlighting here is that, it would enable adaptive and intelligent selection at runtime.

**Comparing FNN-Guided vs Static Decoders:**

Does the adaptive system provide more performance than that of using SC, SCL, or LVA

BER performance and power consumed are analyzed as a function of SNR.

The value of the ML guided approach is quantified in real world.

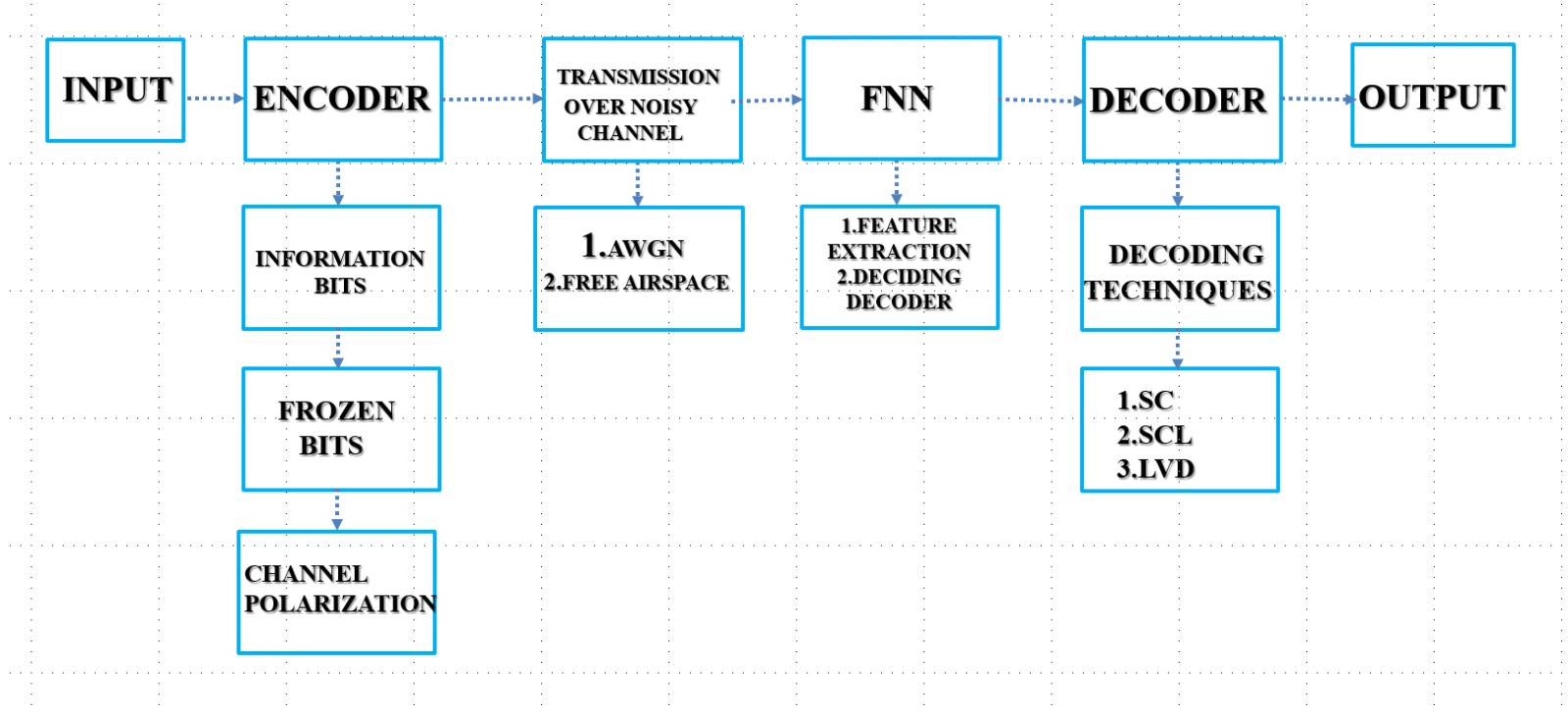
**Block Diagram:**

Fig. 1 Block Diagram

* 1. **Objective**

In this work we initiate a project to intelligently adapt Polar code decoders to resource limited wireless systems. First of all, we take a look into decoding algorithms of Polar code in detail. Three key variants will be implemented, the low complexity Successive Cancellation (SC) decoder, the more powerful Successive Cancellation List (SCL) decoder and most powerful List Viterbi (LV) decoder, as their complexity, respectively, scales as O(1), O(1), O(N). After implementation, we will perform an extensive error correction comparison under various conditions.

For the sake of creating a realistic testing ground, we will simulate wireless data transmission by using Binary Phase Shift Keying (BPSK) modulation. The impairments of a Rayleigh fading channel will be made on this modulated signal, a signal that is subject to the signal fluctuations usual associated with multipath environments. Also, we will incorporate Additive White Gaussian Noise (AWGN) to embody the random noise existing in communication systems. With this simulation environment, we will be able to view the raw performance of Polar codes as well as the effectiveness of our selected decoders under ever tolerant but sensible conditions.

Besides, we go beyond the static decoder selection. For this, we will extract valuable features from the communication process itself. These are the features: number of transmission retries, the rate of packet loss, success of report received from the acknowledgments, and Cyclic Redundancy Check (CRC) errors. These are dynamic metrics of channel quality and success of data recovery.

In parallel, power consumption models for each of the implemented decoder variants will also be developed. To achieve this, we will analyze, in particular, how the computed patterns position themselves and interact, and what their computational and potentially their energy usage, profile is in a practical setting. In order to avail a good trade-off between energy efficiency and services offered, it is imperative to be able to compare the power cost of each decoding algorithm.

Our adaptive system’s intelligence will be a Feedforward Neural Network (FNN). To this end, we will design and train this FNN to predict the most power efficient and reliable decoder given the real time communication features we extract. We feed the network with data that relates channel conditions, decoder performances and the optimal choice in the sense of minimizing error correction and power consumption.

To validate our approach in the real world, we will integrate and test the whole system on the ESP32 microcontroller with LoRa SX1278 radio transceiver, a widely acknowledged LoRa module with long range and low power consumptions. Such a hardware implementation will show the possibility as well as effectiveness of our adaptive decoding strategy in resource constrained devices.

We will finally evaluate the performance of our system with a comprehensive set of metrics rigorously. To quantify the reliability of communication we will measure the Bit Error Rate (BER), for estimating the robustness of the system in adverse conditions, we will measure the outage probability, to characterize the channel we will analyze its Power Delay Profile (PDP), and measure the average power consumption to demonstrate energy efficiency gains due to our adaptive decoder selection.

**1.5 Scope of Work**

*1.5.1 Included in the Project*

#### **Simulation of Polar Encoding and Decoding in Python**

This consists of implementing and simulating Polar codes in Python. The simulation would show the work of the encoding and the decoding processes to provide the data integrity in noisy channel.

#### **Channel Modeling Using Rayleigh Fading and AWGN**

**Rayleigh fading**: It is a model for the signal strength random variation owing to multipath propagation (when signals travel by different paths and interfere with each other). In general, it is used to simulate real wireless communication environments.

**AWGN (Additive White Gaussian Noise)**: An abstract model of random noise added at the output of a signal that is used for simulation. Background noise in a channel is a basic assumption in many communication system simulations.

**Implementation of Decoders**

The project involves implementing three decoding algorithms:

#### **Successive Cancellation (SC) Decoder: Decoder reads bits one by one, canceling the previous one during reading. Polar codes are its basic decoder.**

#### **SC (Successive Cancellation) Decoder: A rudimentary form of error correction using SCL Decoder, which is also kept abreast of multiple possible alternatives for codeword candidates.**

#### **Viterbi Decoder : A variation of the Viterbi algorithm in which it does not only track the most likely path, but also tracks all possible paths to improve the decoding accuracy.**

#### **Dataset Generation Using Performance Features**

This step entails generating a dataset that characterises the performance of the different decoders in different conditions. The possible features could be BER, SNR, and other related performance metrics. The evaluation indicates how well the decoders perform in realistic settings, on this dataset.

#### **FNN (Feedforward Neural Network) Model Training and Inference Integration**

In this phase, a neural network or a feedforward neural network is trained over the generated dataset (from previous phase). This model is then integrated into the decoding process to help the error correction. This is on the proposition that it is possible, through machine learning techniques, to work more efficiently than traditional methods in decoding.

#### **Power Modeling and SNR-Aware Adjustments**

It is a process of modeling the power consumption of the system and making adjustments to maximize system performance depending on the SNR (Signal-to–Noise Ratio). Due to the battery limited operation of the system, there is a need to conduct power management to prolong the operation of the device, while at the same time achieving the best possible communication performance.

#### **Hardware Deployment on ESP32 with LoRa Module**

After developing the simulation and algorithms, they will be deployed on the actual hardware. The LoRa module allows a long range of communication, and the ESP32 is a low cost microcontroller with built in Wi-Fi and Bluetooth. This hardware setup will also enable real world testing of the Polar encoding and decoding system in a communication environment where the error correction and efficient transmission algorithms designed will be used.

*1.5.2 Excluded from the Project*

**Physical Layer MAC Protocol Design**

In other words, this is about designing the Medium Access Control (MAC) protocol at the physical layer that controls data transmission scheduling, providing access to the communication medium, and managing the network contention. It is not part of the project, i.e. its main purpose is to correct error and channel coding, and not network access protocols.

**Real-time Adaptive Modulation Schemes**

The project does not include the dynamic (i.e., in the run time) adjustment of modulation schemes (i.e., QPSK, QAM) according to the changes in the network conditions, e.g. SNR. However, this technique permits the system optimize the throughput but is outside the current scope.

**Online Learning or Real-time Training Updates**

This does not include the case of real time model update (or online learning) wherein the system can adapt and learn in an online manner during operation by incorporating new data into its model. The project is about pre trained models and does not cover continuous leaning from deployment onwards.

**Integration with Cloud-Based Backend Systems**

For this we mean that, since the project won’t include linking this system towards cloud infrastructure for uplinking data for underwriting purposes, processing and data analytics, the electronic system need not include this. We focus on local processing and on the deployment on hardware without interactions with the cloud.

**1.6 Research Questions**

This study aims to answer the following key questions:

### **Decoder Selection Accuracy**

How accurately can an FNN choose the optimal decoder at real-time given a clear link layer feature (e.g. measured SNR or BER)? The second type of reinforcement is the only one that enables the FNN to select which decoding algorithm (SC, SCL, LV) is the most adapted under different conditions.

### **Performance Trade-offs**

The BER and power consumption of FNN based decoder selection is compared to static decoders. Therefore, the FNN should choose simpler decoders for low power and more complex decoders for required high accuracy.

### **Hardware Feasibility**

Can the FNN and decoders be run on a constrained device such as the ESP32? Because of the ESP32’s limited resources of the ESP32, the problem is to run the FNN efficiently along with the decoding algorithms while not overloading its processing capabilities.

### **Power Efficiency vs Performance**

Is it possible to realize SCL/LV performance with SC power dissipation? The desired properties of FNN are that it chooses decoders that keep high performance while avoiding any unnecessary power consumption, thus energy efficient and error free.

**1.7 Methodology Overview**

*1.7.1 Polar Code Construction*

Let be the block length, and KKK the number of information bits. The encoding is done using a generator matrix**,** where:

* F =
* denotes the n-th Kronecker power of F
* is the bit-reversal permutation matrix

The encoded vector is:

Where with frozen bits set to 0 based on Bhattacharyya parameter analysis.

*1.7.2 Channel Modeling*

The encoded message is modulated using Binary Phase Shift Keying (BPSK):

The modulated signal is passed through a Rayleigh fading + AWGN channel. The received signal y is modeled as:

Where:

* h∼R(1) : Rayleigh fading gain
* : Gaussian noise
* x : Transmitted BPSK signal

*1.7.3 Decoder Algorithms*

* **SC**: Low-complexity sequential decoding.
* **SCL**: Maintains a list of candidates with path metrics.
* **LVA**: Performs likelihood-maximizing search over code tree.

*1.7.4 Feature Engineering and Dataset Generation*

We simulate communication across SNR levels and extract the following features:

* **Retries**
* **Packet Loss**
* **ACK Success Rate**
* **CRC Error**

These are used to label the best decoder (lowest BER) for supervised training of the FNN.

*1.7.5 Neural Network Training*

A feedforward neural network is trained using scaled features. Output is a softmax over decoder classes:

*1.7.6 Hardware Profiling and Power Modeling*

Power is estimated as:

Where computation power depends on the number of instructions (decoder complexity) and SNR-driven dynamic scaling:

* SC: ~10,000 instructions
* SCL: ~50,000 instructions
* LVA: ~100,000 instructions

**1.8 Thesis Organization**

* **Chapter 1** provides an overview, motivation, objectives, and methodology.
* **Chapter 2** covers the literature review and related work.
* **Chapter 3** details the system architecture and decoding logic.
* **Chapter 4** explains the simulation setup, tools, and evaluation metrics.
* **Chapter 5** presents results and performance analysis.
* **Chapter 6** discusses the hardware implementation.
* **Chapter 7** concludes the thesis and outlines future work.
* The **Appendices** include code snippets, architecture diagrams, and datasets.

**CHAPTER 2: LITERATURE REVIEW**

**2.1 Introduction to Channel Coding**

Digital communication system uses channel coding as it increases the ability of a system to detect and repair errors introduced as a result of noise or other impairments in the channel of communication. Channel coding is primarily for ensuring that the receiver can recover the original message in a reliable fashion even in the event of a part of the signal being corrupted in the transmission.

### **How Channel Coding Works**

### **This is because, in the process of transmitting information, the raw data (commonly referred to as source message) is prone to errors caused by noise, interference or other distortion in the communication channel. To overcome this problem, channel coding increases redundancy in the data to be transmitted across the channel. The redundancy is considered, in detail, so that it will enable the receiver to detect and possible correct errors that may occur during the transmission.**

### **Encoding Process**

Channel coding is the process whereby the source message, uuu (or the input message), is mathematically altered into a longer word, xxx. The encoding will introduce redundancy, which the receiver should be able to see some kind of error and fix it.

We do this transformation with the help of a generator matrix GGG. If uuu is the input message, the encoded message xxx is the product of the input message uuu multiplied by the generator matrix GGG

Where:

* : is the input message vector, typically a sequence of binary values (0 or 1)
* : is the encoded message (or codeword) that will be transmitted over the channel.
* : is the generator matrix, which defines how the input message is mapped to the encoded message. It contains the rules for adding redundancy to the original message.

**2.2 Evolution of Error Correction Codes**

Over the years, different generations of error correction codes have been proposed and they overcame the limitations of previous generations in terms of efficiency of detection and correction of errors.

### **Hamming Codes**

### **It is one of the earliest block codes, which can correct single bit error. However, those are simple and effective in low error scenarios, but limited in performance in noisy environments.**

### **Convolutional Codes**

### **The introduction of memory in the codes allowed better error detection and correction over sequences of bits, instead of individual blocks.**

### **Turbo Code**

### **Turbo codes introduced in the 97 were introduced, iterative decoding, and come very close to the Shannon limit: the theoretical maximum data rate for error free communication.**

### **LDPC Codes**

### **Sparse matrices and graph based message passing algorithms form the basis of the decoding in terms of low complex performance code in Low Density Parity Check codes that are very suited to high throughput applications such as 5G and Wi-Fi.**

### **Polar Codes**

### **First, these are the only known codes that are proven to yield the channel capacity of symmetric B-DMCs as the block length N→∞.**

### **Channel Capacity and Its Importance**

Communication channel capacity CCC is the maximum rate of transmission of data on a communication channel with negligible error. In the case of a binary input Additive White Gaussian Noise (AWGN) channel it is given by:

**.**

Where **SNR** is the signal-to-noise ratio. This formula represents the theoretical upper limit for

reliable communication.

**2.3 Polar Codes and Their Importance in 5G**

During 2009, Erdal Arıkan introduced Polar codes which are rightly called a major milestone in the field of error correction coding. The core of their innovation is channel polarization. In this process, a set of identical binary input channels are transformed to a new set of bit channels that interleave bit errors, some of them are in highly error prone channels while the other are in highly reliable channels. In transmission, the information bits are mapped to the most reliable bit- channels (for accurate delivery) whereas the frozen bits (that are preset with known values (usually 0’s) and are known by both encoder and decoder) are mapped to the unreliable channels in order to help in the decoding process.

The encoding operation is defined as:

The encoding process of Polar Codes is mathematically defined using a **generator matrix** GNG\_NGN​, which is formed by the product of a **bit-reversal permutation matrix** BNB\_NBN​ and the **Kronecker power** of a base kernel matrix FFF. Specifically, the formula is

* F =
* : The Kronecker power recursively constructs a larger transformation matrix from the basic kerne
* ​: Reordes the bits to align them for polarization
* u: Input vector with frozen bits inserted

**Why 5G uses Polar Codes**

On the other hand, the use of Polar codes in 5G systems was mainly due to their superior performance in short packet communication over noisy wireless channels. Unlike Turbo codes, which were very popular in 4G, Polar Codes can better correct errors in the short block lengths of which control channels often consist of, where fast and reliable data signaling is critical. Their selection as the best candidate is in the part due to the low complexity Successive Cancellation (SC) decoding algorithm, which makes Polar Codes inexpensive to implement on hardware with low resources. Moreover, Polar Codes use adaptive decoding methods such as Successive Cancellation List (SCL) decoding that enhances the performance by carrying out multiple decoding paths and hence takes better decisions when conditions are unclear. This flexibility offers us a tradeoff between the system complexity and reliability. Nevertheless, due to these advantages, Polar Codes are favored primarily for the control channels in 5G applications such as enhanced Mobile Broadband (eMBB) and Ultra- Reliable Low-Latency Communications (URLLC) which require high reliability and low latency.

**2.4 Review of Polar Decoding Techniques**

*2.4.1 Successive Cancellation (SC)*

Standard decoding method for the Polar Codes is the Successive Cancellation (SC) decoder. It is a sequential decoder that starts decoding each bit sequentially one by one from the first to the last using log-likelihood ratios (LLRs) to make its decisions regarding the received signal. Given the measured output of the channel, so far identified as letters from an alphabet, the LLR tells the MT something about the probability that the channel output is more likely to represent a 0 or a 1.

The LLR for a received symbol y is defined mathematically as:

This formula is a ratio of the probability that yyy was received if the transmitted bit was 0 versus if it was 1. If the LLR is positive, the bit is more likely to be 0, if the LLR is negative, it is more likely to be 1. The LLR values can be used by the SC decoder to estimate bits one by one with consideration to earlier bit decisions and the code structure. While efficient, this is sometimes not what you want, especially if jitter, noise gets in the way, in which case, enhanced decoders such as SCL are used to beat up and perform better

The successive cancellation (SC) decoding of Polar Codes is performed recursively at binary tree structure in which every node corresponds to a smaller sub code. To achieve this, we need a rule combining LLR of left and right child nodes to compute the LLRs (Log-Likelihood Ratios) at each node. It is essential because it allows one to propagate reliability information gradually up the decoding tree.

The exact LLR calculation is a simplified version of the “min-sum approximation”.

When the decoder traverses the binary decoding tree in Successive Cancellation (SC) decoding of Polar Codes, it visits the left and right child nodes at the time of recursive processing. After the left child (even index) is decoded and its estimated bit is known, the decoder computes the LLR for the right child (odd index).

By this formula, LLR of the right child is calculated.

Where:

* û is the previously decoded bit.

### **Advantages:**

* **Low complexity: O(N log N)**

Consequently, for algorithmic time complexity, its order of growth is proportional to N log N for the size of the data or problem, N. This is regarded as efficient in the case of larger datasets or systems.

With the logarithmic term, the algorithm is not really that slow in the case of larger problem size, useful for the systems with low computational resources, for example, an embedded system.

* **Suitable for embedded systems:**

In fact, the processing power, memory and battery life of embedded systems are typically quite limited. This algorithm is efficient in that its complexity is O(NlogN), enough for it to run on such systems. This is appropriate for usage in devices with constrained resources, yet high performance is required.

### **Limitations:**

* **Poor BER performance at high SNR (Signal-to-Noise Ratio)**

Bit Error Rate is the abbreviation for BER, and it is a measure of how many of the total bits received are received incorrectly (due to noise or interference).

In the case of high SNR, the signal strength is much higher than noise and thus, it is easier to pick the correct bits from error.

Poor BER performance at high SNR refers to the condition where, even with ideal signal reception (low noise and high signal), an algorithm or decoding method is not able to decode the information with a high level of accuracy. The reason could be that the algorithm is not designed to handle such conditions optimally.

* **High decoding latency due to its sequential nature**

In other words, decoding latency is the amount of time consumed in decoding the received data.

If the algorithm has sequential nature, then the data are processed one step at a time, and not parallel. However, this creates delays when the data size expands.

While latency is high, in scenarios where speed is critical (e.g., in case of real time communication) it is a drawback because it would take long time for the system to decode and process the information.

*2.4.2 Successive Cancellation List (SCL)*

The Successive Cancellation (SC) decoding of polar codes can be enhanced by using Successive Cancellation List (SCL) decoding. In SC mode, decoding process one decoding path sequentially; however, in SCL mode, it keeps multiple decoding paths for each decoding step. Decoding is not performed for a specific candidate decoding only, but explores several, what allows having more flexibility and robustness.

The PM is the key aspect of SCL decoding. The path metric is this and based on this it helps to determine how much chances it has for each path to be correct codeword. The updated path metric for each decoding step is taken as

L in this case, is the Log-Likelihood Ratio (LLR) value of that specific decoding step that conveys the confidence in that received bit. The higher the value of the LLR, the higher the confidence in the value of the bit; on the other hand, a low value (or negative value) of the LLR constitutes lower confidence.The value log(1 + e^{-L} ) is used for updating the path metric on the basis of the existing value of LLR. That way paths with a greater confidence (higher LLR value) are updated less in their path metric and are less likely paths (smaller LLR value) are penalized more. SCL decoding keeps separate paths and continually updates those metrics on how likely each path is, and thus allows for more paths to be explored and fewer errors than SC decoding is prone to in noisy or imperfect conditions. Such a scheme improves performance, particularly regarding error correction, but at the expense of increased complexity due to handling multiple paths; SCL selects the most likely path upon decoding and will frequently compose a CRC check to verify the output is correct.

**Advantages**:

* **Near-Optimal Decoding Performance (with CRC aid)**

SCL decoding coupled with Cyclic Redundancy Check (CRC) provides good performance up to the theoretical limit for error correction of polar codes. This is sometimes called ‘near optimal’ performance.

**Benefit:** SCL is very well at decoding correctly even pretty close to what would be considered the best possible decoder to use (Maximum Likelihood Decoding). CRC also helps by checking the validity of the decoded output and gives very accurate and reliable final result.

* **Improved Error Correction in Noisy Channels**

SCL decoding offers a drastic improvement in comparison to SC decoding, especially when there is signal degradation or interference during communication so the received data is erroneous.

**Benefit :** SC decoding decodes one path at a time, resulting errors if noise or the interferences act on the first steps decoding. However, SCL performs at the cost of maintaining multiple paths so that the decoder is able to explore different possible solution. Additionally, by doing this, the BER in noisy conditions is greatly reduced.

* **Increased Flexibility by Maintaining Multiple Paths**

SCL decoding maintains several paths while tracking their respective scores, which are based on the decoding likelihood. This flexibility enables further searching for possible codewords.

The benefit is that SCL attempts to evaluate multiple possible solutions in parallel, which if the code is to be decoded with minimum error under noisy channels, multiple solutions must be evaluated in parallel to increase the likelihood of choosing the correct codeword. This cuts down on a problem with SC decoding where the decoder can get ‘stuck’ on an incorrect path.

**Disadvantages**

### **Higher Complexity and Memory Use: O(N log N)**

SCL decoding requires more computational power and memory to maintain multiple paths and their metrics, resulting in higher complexity and memory usage compared to SC decoding.

### **More Demanding for Low-Power Devices**

The increased complexity and memory requirements make SCL decoding more power-hungry, which can be problematic for low-power or battery-constrained devices.

*2.4.3 List Viterbi Algorithm (LVA)*

However, the traditional Viterbi algorithm is an extension of algorithm to only return the most probable path, whereas the List Viterbi Algorithm (LVA) returns multiple paths to improve the error correction. In traditional Viterbi decoding, the algorithm keeps iterating through possible states and always selects the single most probable path, but when the received signal contains uncertainties or errors, such single most probable path becomes limiting.

To address this limitation, LVA keeps tracks of the most likely paths throughout the decoding process. This makes the algorithm more robust to noise since now it is possible to search bigger region of potential sequences for the potential decoded sequence that is the most likely the original transmitted codeword.

In LVA, the probability of each decoding path is evaluated and compared according to their path metric (M). It is calculated according to the formula.

Where:

* represents the received symbol (at the iii-th position),
* represents the corresponding transmitted symbol (at the iii-th position),
* is the likelihood of observing yi given the transmitted symbol

Yi is the likelihood of receiving a particular signal Yi given that the transmitted signal was Xi . When AWGN channels exist, this means that noise has been added to signal transmitted such that it affects the received signal Yi. The noise has a Gaussian distribution with zero mean and variance

The formula for the conditional probability is given as

**Advantages**:

* **Best BER Among the Three Decoders**

**Description**: The performance of LVA gives the best Bit Error Rate (BER) among the other decoding algorithms which include Successive Cancellation (SC), and Successive Cancellation List (SCL). This is because LVA maintains several decoding paths and selects the most appropriate one that could be the optimal compared to OCC, resulting in a higher likelihood of finding the proper codeword despite noise or error in the channels. LVA is able to explore more potential solutions, and so corrects more errors, resulting in better performance in terms of BER.

**Benefit**: Due to its error correction we can obtain better performance in noisy environments such high noise or signal degradation (for example in wireless communication over AWGN channels).

* **Works Well with Short Codes**

**Description:** LVA works especially well in the case when the codes we are trying to decode are short codes, that is, there is a relatively small set of possible decoding paths. LVA remains therefore more suitable when the code length (number of paths) is small, since it maintains multi paths.

**Benefit**: Short codes are often used in low latency or real time system where the decoding overheads must be kept to a minimum. For these cases, LVA works very well, it helps correct error without inducing major performance penalty.

**Disadvantages**:

* **High Computational Burden**

**Description**: Profiling and the list chosen for this Viterbi algorithm is bound to consume a lot of computational resources as it maintains and processes multiple decoding paths at the same time. In each step of the decoding process it has to calculate the path metric of all candidate paths, which is more arithmetic intensive than SC, or even SCL.

**Why it’s a disadvantage**:Herein, as more paths are taken, the computational load is also exponentially increased. Due to this, LVA becomes computationally expensive, particularly for long codes or when a big number of paths is requested. This requires a high processing power which can translate to longer decoding times and increased number of calculations.

* **Not Practical for Battery-Operated Embedded Devices**

**Description:** The growth of the computational load is exponential with the number of paths. Due to this, LVA becomes computationally expensive, particularly for long codes or when a big number of paths is requested. This requires a high processing power which can translate to longer decoding times and increased number of calculations.

**Why it’s a disadvantage:** Often, the battery powered IoT sensors, mobile devices and wireless communication modules have strict energy consumption limitation. However, the computational burden of LVA is large enough that it would result in excessive power consumption and decreased battery life, making it impractical for use in real time or for long periods in real time in an embedded system.

**2.5 Neural Networks in Communication Systems**

Recently, artificial intelligence (AI) especially the neural network is integrated into the communication systems in a significant change from traditional model based signal processing approach to a dynamic data driven one. Traditional communication equipment relies on fixed mathematical models and sets of algorithms designed to work optimally under certain, normally assumed, conditions. However, these models have worked well in many applications, but it can be quite hard to use them in the real world where channel conditions are dynamic, noise is varying, the interference patterns are unpredicted.

Particularly for the complexity associated with neural networks from the ML world, AI in general provides a more flexible and adaptive approach for dealing with these complexities. Through training on data, neural networks have the aptitude to change their logical decisions with respect to varying situations, something that conventional approaches are not capable of. Neural networks are capable of processing a lot of data, adapting to new patterns, and performing real time decisions which makes optimal for situations where constants are changing frequently.

For the purposes of communication systems, this movement has manifested itself in that, previously, signal processing algorithms such as modulation, demodulation, and decoding were being improved or replaced by machine learning models. These models can learn from the data they receive and autonomously optimize how they process and decode information, which gives a better performance for a system, particularly in either complex or noisy environments.

.

*2.5.1 Applications of Neural Networks in Physical Layer Tasks*

Adaptive data driven solutions are provided by neural networks, specifically Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs), to overcome the problems in communication systems with great success. The advantages of the MFDS in core areas of the communication system are explained below more in details:

### **Channel Estimation**:

**Both CNNs and fully connected networks are trained as neural networks which can learn some complex relationships between transmitted pilot signals and Channel State Information (CSI). Such channel estimation is of critical importance to estimate the channel conditions, which can possibly include different degrees of multipath fading, i.e., the presence of multiple copies of the signal reaching the receiver via reflections and scattering, or non-linear distortions, such as those induced by hardware imperfections and environment parameters.**

**Advantage**: In fact, traditional approaches of channel estimation are often based on some traditional rigid mathematical models, which may fail in complex, real world channels. However, neural networks can overcome such complexities through the learning on the observed data, which makes them effective in predicting the channel state accurately in real‐hard conditions. As a result, the overall system performance is improved and signal reception is strengthened.

* **Signal Detection**:

**In the traditional communication systems, signal detection is achieved by demodulating the received symbol by using predefined algorithms which estimates the behavior of the channel. However, the rule based systems do not have good performance for non linear or highly dynamic channel conditions.**

**DNN based detectors detect and demodulate symbols by estimating and hence learning from the statistical and non linear behavior of the communication channel. Neural networks also are different from conventional methods in that they make no particular assumptions about the channel and so are more accurate in the detection of signals in noisy and unpredictable channels.**

**Advantage**: The ability of DNN to identify complex pattern of data and detect with high accuracy signal even in environment with interference or fading, improve a great deal their reliability to detect and reduce error rate.

* **Modulation Classification**

The task of modulation classification is to determine which of these modulation schemes (BPSK, QAM, FSK, etc.) is used to send the signal contained therein. Typically, it entails the analysis of signal features followed by the use of rule based techniques. However, classifying modulation scheme is difficult especially in the presence of noise or signal distortions.

CNNs and hybrid networks in particular are particularly well suited to this task, as they are able to perform raw I/Q image of samples (In-phase/Quadrature phase data) or spectrograms of automatically classifying the modulation scheme. In particular, spectrum surveillance and cognitive radio are the application areas where the goal is to detect and use frequency bands without knowing a priori what the transmitted signal is.

**Advantage**: The modulation type can be efficiently classified using neural networks without having to explicitly extract the features, resulting in faster, more accurate, and noise and distortion robust classification.

### **Decoding**

**By adapting belief propagation (BP) for low‐density parity‐check (LDPC) conclusion or progressive accumulation (SC) for Polar codes, neural decoders have demonstrated the potential to reprise or exceed these meeting as of now known iterative decoding calculations. However, such as algorithms are widely used in modern communication systems, yet, they suffer from disadvantages such as high computational complexity or long decoding latency.**

**But training such a neural decoder on data about these algorithms forces it to learn the underlying structure of them and then can do the decoding task much more efficiently — often with lower latency and in some cases more accurately. For example, neural networks can adaptively choose the appropriate decoding strategy based on real time environment, which reduces the computational cost and time of the system compared to the existing methods.**

**Advantage**: Neural decoders that reduce the loss in time for decoding and may improve performance are advantageous to take advantage of in real time communication systems, where speed and accuracy are crucial.

*2.5.2 Feedforward Neural Network for Decoder Selection*

For that specific project, we were using a Feedforward Neural Network (FNN) to make an intelligent decision as to which procedure to use for decoding the signal, based on the extracted signal features. Communication systems usually resort to given decoding algorithms, e.g., as Successive Cancellation (SC) or Viterbi. Algorithms are developed for automatic parameter estimation but their performance can depend on signal quality, noise level and interferences.

An FNN is a particular kind of artificial neural network whose data moves unidirectionally i.e., from the input to the output through a bunch of processing units (neurons). The input features (in general, signal characteristics, noise level and channel conditions) are mapped by the network to an output (an optimal decoding algorithm). To train neural network, we feed it with data and it learns which decoder is suitable for certain given conditions.

This advantage lies with the fact that the FNN does not depend on fixed assumptions about the signal or the channel, but learns from the actual signal characteristics, so that it is much more adaptive. By training the FNN on a number of signal scenarios, it is able to determine realtime which decoder is best, thus improving the overall performance of the communication system in regards to error rate, efficiency and robustness.

In essence, it utilizes neural network technology as a means to automate and optimize decoding to achieve the intelligent decision making capability on different conditions without manual intervention or pre-defined models. Using this approach, communication system can be made more efficient, robust and adaptive, especially in the face of highly variable and complex environmentsModel Architecture

An Artificial Neural Network that is the simplest form and does not form any cycles between the nodes or connections of the network is called a Feedforward Neural Network. The model has the form of an input layer followed by one or more hidden layers and ends with an output layer. A non-linear activation is followed by a linear transformation in each layer.

* **Input:** The FNN takes in a feature vector , where each dimension corresponds to a signal characteristic. In our case, features include:

1. **SNR (dB)**
2. **Modulation type**
3. **Channel State Information (CSI)**
4. **Mean Log-Likelihood Ratio (LLR)**

* **Hidden Layers:** Each hidden layer computes a non-linear transformation of its input and gives output:

The final prediction is given by the softmax function:

Where:

* x is the input feature vector,
* is the score for class iii,
* W, b are the weight matrix and bias.

This allows mapping from signal conditions (retries, packet loss, etc.) to the optimal decoder type.

*2.5.3 Training the FNN*

A large dataset comprising over 10 thousand labeled examples was created in order to effectively train the Feedforward Neural Network (FNN) for intelligent decoder selection. The training examples consist of a feature vector describing the main characteristics of the signal (SNR, noise variance, channel type, etc.) and a label defining which decoder is the best to use for the given conditions. Because this is a supervised learning setup, the FNN is able to learn the mapping from signal conditions and optimal decoder choice.

**Label Generation:**

Empirical evaluation process was used to generate labels used for training. All three decoder types, namely Successive Cancellation (SC), Successive Cancellation List (SCL), and List Viterbi Algorithm (LV), were applied to the received data for each signal scenario in the dataset. They were assessed in terms of bit error rate (BER) and energy consumption using defined threshold or benchmarks.

* For a given input we selected it’s ground truth label as the label from the decoder with the least BER.

Alternatively, for minimizing energy (e.g., to reduce power consumption in low power devices), the decoder with the lowest energy usage under acceptable BER limits was selected.

Performing such a comparative evaluation ensured that the labels represent the decoder that would actually be optimal in practice and hence the neural network can learn a decision boundary that generalizes without seeing the test conditions.

**Loss Function**:

In order to train the FNN, categorical cross entropy loss was chosen as the category is a multi class classification task. The comparison of the network’s predicted probability distribution over the three decoder options to the true label (optimal decoder) and penalization of strongly held incorrect beliefs the more confident the network is about the incorrect answer.

The loss for a single example is mathematically:

* Where y ∈{0,1} is the one-hot encoded label for the correct decoder.

**Optimization**

Feedforward Neural Network (FNN) was trained with Adam optimizer, which combines effective of adaptive learning rate and momentum of speeding up the convergence and processing the sparsely gradient. Weights were updated using small subsets of the data in Mini batch gradient descent to keep a training speed vs stability balance. Early stopping was utilized to terminate the training at a point of no improvement in the validation performance, dropout was adopted to turn some neurons on or off randomly during training to avoid having the model overly dependent on a particular set of nodes, and L2 regularization was utilized to penalize large values of the weights and hence favor simpler and more resilient models.

*2.5.4 Advantages of Using FNN for Decoder Decision*

Feedforward Neural Network (FNN) can be used to select decoder at inference time with significant benefits as low latency inference. Eventually the model is trained and can make quick decisions with very little overhead. This enables the features, such as in wireless communication or IoT systems, to be quickly processed and the best decoding strategy to be chosen, in real time without delay, with an efficient and responsive system.

Another significant advantage is that the FNN has the adaptive intelligence. The neural network is not a rule based technique and can generalize over many different channel conditions and noise levels. Without manual parameter tuning or pre-set conditions, it dynamically adapts to the current environment so as to select the most fit decoder. However, this makes the communication system robust and capable of achieving the same performance in a highly unpredictable environment.

The energy optimization is also contributed by the FNN. It allows for analysing signal quality and prefer energy-efficient decoders, SC for beneficial channels and LV for poor channels. The selection of this targeted decoder reduces power consumption, an important optimization when using battery powered devices or devices with limits placed on energy consumption.

Lastly, FNNs are hardware friendly and good for use on embedded platforms like ESP32 and ARM based microcontroller. This flux in artificial intelligence has been excellent for small sensors and ocean hardware due to the ability to train FNNs and easily incorporate them into friendly frameworks such as TensorFlow Lite and MicroPython on compact hardware with no powerful processor or cloud access needed to make intelligent decisions at the edge.

## ****2.6 Energy Constraints in IoT Devices****

In low power wireless communication systems, for example remote sensing, IoT, telemetry, ESP32 and LoRa SX1278 are frequently used devices. These devices usually rely on batteries power supply, and thus it is important to reduce energy consumption in order to prolong operational lifetime, as well as to reduce mechanical maintenance.

For such systems, the total power consumed can be represented as the sum of two parts:

Where:

* **:** represents the **constant power** used by the radio receiver during communication. For example, the LoRa SX1278 has a fixed receive mode power that remains consistent regardless of processing demands.
* : accounts for the power consumed by the microcontroller (e.g., ESP32) to execute the decoding algorithm. This component varies based on the **number of instructions** required by the chosen decoder.

Computation power:

The above formula explains that the more complex the decoding algorithm (such as SCL or LV), the more instructions are needed and thus result in higher energy use. On the other hand, simpler decoders as SC need to consume fewer instructions and consequently less power, which requires less transistors. Such a trade-off demonstrates the necessity for the intelligent choice of a decoder that provides optimal performance while being energy efficient primarily on constrained devices.

In our implementation:

* SC ≈ 10,000 instructions,
* SCL ≈ 50,000 instructions,
* LVA ≈ 100,000 instructions.

This forms the basis of our decoder selection strategy to choose the most power-efficient decoder for a given signal condition.

## ****2.7 Existing Works on Decoder Selection****

Many strategies to adaptively select the best decoding algorithm for the communication environment have been developed. An SNR thresholding approach is one of the simplest, which consists of applying a single rule: when the SNR is above a certain threshold, a lightweight decoder like SC is used, otherwise a more powerful decoder like SCL or LVA is used. Look up tables based on static mappings between observed channel metrics (e.g. noise level or SNR) and the considered decoder choice provide a fast but inflexible decision making.

Sophisticated methods make use of machine learning models from the classes of classifiers and regressors to predict the expected decoder performance behavior under changing conditions. It is possible to train these models on historical data and solve many unseen scenarios well. Further, some researchers have studied cross layer optimization, where decoding strategy is changed together with medium access control (MAC) layer parameters so as to achieve the system wide efficiency across physical and network layers.

However, many of these strategies are limited in various ways. Usually, they pay no attention to practical, real world metrics like packet retries, ACK success rate or latency that actually make a difference in real communications in deployed systems. Besides, certain approaches, especially ones in which complicated ML models or synchronous layer adventures are taken part, are excessively computationally expensive making them impracticable for use on low profile embedded devices such as ESP32. Finally, several of the presented strategies fail to appropriately consider power consumption, which is a key concern for battery powered systems, and consequently must be considered by any decoder adaptation strategy that wishes to be utilized for real world deployment.

## ****2.8 Research Gap and Contribution****

Despite extensive research on Polar Codes and decoder optimization, existing work fails to address **real-time adaptive decoder selection in battery-powered embedded devices** using link-layer metrics.

**Our contributions include**:

Many strategies to adaptively select the best decoding algorithm for the communication environment have been developed. An SNR thresholding approach is one of the simplest, which consists of applying a single rule: when the SNR is above a certain threshold, a lightweight decoder like SC is used, otherwise a more powerful decoder like SCL or LVA is used. Look up tables based on static mappings between observed channel metrics (e.g. noise level or SNR) and the considered decoder choice provide a fast but inflexible decision making.

Sophisticated methods employ machine learning models (classifiers or regressors) to predict expected performance of decoders with respect to the operating conditions. It is possible to train these models on historical data and solve many unseen scenarios well. Further, some researchers have studied cross layer optimization, where decoding strategy is changed together with medium access control (MAC) layer parameters so as to achieve the system wide efficiency across physical and network layers.

Yet, many of these strategies are hampered significantly. Usually, they pay no attention to practical, real world metrics like packet retries, ACK success rate or latency that actually make a difference in real communications in deployed systems. Besides, certain approaches, especially ones in which complicated ML models or synchronous layer adventures are taken part, are excessively computationally expensive making them impracticable for use on low profile embedded devices such as ESP32. Finally, several of the presented strategies fail to appropriately consider power consumption, which is a key concern for battery powered systems, and consequently must be considered by any decoder adaptation strategy that wishes to be utilized for real world deployment..

**CHAPTER 3: SYSTEM ARCHITECTURE & METHODOLOGY**

**3.1 Overview of the Proposed System**

In this project, we try to optimize the Polar Code decoding to conduct on energy constrained embedded systems with the help of a hybrid approach that merges the classical decoding algorithms (SC, SCL, and LVA) and modern machine learning techniques to select the decoder in real time. In this system, real time feedback features of the communication link are extracted and used by the system to dynamically choose the most appropriate decoding method to use for a given link condition. Because of this adaptability, the decoder complexity will be minimized when the signal is strong and hence will save energy, while ensuring that performance will be high when the signal is poor, balancing performance and energy consumption.

The system architecture consists of three major layers for which each layer has its own unique role in fulfilling the project objective:

**Simulation Layer**

The bottom layer, written in Python, is used to implement a controlled environment to test and evaluate the system performance. An end to end, encoded message is transmitted using Rayleigh fading and AWGN (Additive White Gaussian Noise) channel. However, this simulation will be very important for testing the decoding algorithms (SC, SCL and LVA) under different conditions and provides the needed data for training the machine learning model. The simulation helps to understand in what way the system behaves in real world environments by adjusting the channel conditions before implementation on reality hardware.

**Machine Learning Layer**

A Feedforward Neural Network (FNN), which is trained to predict the best decoding method for each received signal is the employed machine learning layer. It is predicted on the basis of various statistical communication features, including SNR, packet retries, ACK success, and CRC errors extracted from the ongoing communication link. These features are evaluated once they are trained on the FNN, and the decoding strategy (SC, SCL, or LVA) that is expected to reach the best performance given energy consumption and decoding accuracy is chosen. This approach enables the system to handle a large variety of channel conditions very efficiently.

**Embedded Layer**

The logic is implemented on real world (hardware) for validation and testing in the embedded layer. There is a LoRa SX1278 module for wireless communication and an ESP32 microcontroller as the processing unit. Its task is to execute the FNN model in real time, to choose the best decoder that can be run in real time setting and transmit and receive encoded messages. The system can interface with the LoRa module to validate the live process of the decoder selection, and give a real picture of the performance of the system in real world, mainly in terms of power consumption, reliability and energy efficiency.

**3.2 Communication Model**

It simulates a real world low power wireless communication case that has energy efficiency and reliable data transmission in various channel conditions. Network management is based on the use of Polar Codes together with the Binary Phase Shift Keying (BPSK) modulation, and this constitutes the communication chain at the core. Both software and hardware layers implement this chain so that the end product could be tested and validated in simulated environments before deploying it to real hardware.

**Polar-encoded BPSK Communication Chain:**

Polar Code encoding for efficient transmission of data through a noisy communication channel forms the leg of communication chain. Error correcting code, called Polar Codes, are known for their capability of approaching the Shannon limit for an increasing block length. In particular, they are particularly suitable for low-latency and low-complexity decoding and thus are suitable for embedded, powerconstrained systems. Before the modulation, the data is encoded into a polar code to achieve high transmission reliability.

The simplicity, low energy consumption are the reasons for selecting BPSK (Binary Phase Shift Keying) modulation for this system. BPSK uses two distinct values, or phase shifts, of information, 0 and 1. Due to the good balance of power efficiency and error performance in noisy environments, BPSK is widely used in wireless communication systems.

**Software Layer:**

The Polar encoding and BPSK modulation are modeled and tested in a software environment in simulation layer. The simulation models the end-to-end communication process such as encoding, modulation, transmission of Rayleigh fading and AWGN channel, followed by decoding, and are carried out in Python. By this, it is possible to examine how different decoding algorithms (SC, SCL, LVA) perform in various situations adopting additonal sources of randomness (noise and fading) during transmission. Most importantly, the software provides the capability to fine tune the machine learning model to pick the decoder for the specific signal conditions.

**Hardware Layer:**

The communication chain of the embedded layer is implemented on actual hardware which includes ESP32 microcontroller for processing and LoRa SX1278 module for communication. The LoRa module is used to communicate with the other devices while the Polar encoding and BPSK modulation are emulated in software on the microcontroller. The hardware system is transmitting and receiving data in real time, in the spirit of mimicking real wireless communication conditions. The second use of the hardware layer is to validate the decoder selection model (trained FNN) so that the system act as it should, in terms of both energy consumption and reliability.

*3.2.1 Message Generation and Encoding*

For this system, the experimental setup of the project is that a pseudo random binary message of length K is fed to the system which then encodes it into a Polar codeword of length N = K\*2^n. Pseudo random generation is used to guarantee messages remain unknown and not fixed pattern or dictionary based payloads so that they there are exposed to a large number of bit permutations. This makes the decoding performance evaluation more complete as you evaluate the system with different bit sequences and consequently also different bit correlations of the input bits.

There is the construction of the frozen bit vector using Bhattacharyya parameters:

Firstly, the Polar Code encoder includes the generation of a frozen bit vector, which is used throughout the encoding process. Frozen bits are bits are whose value has been predetermined (typically 0) and are unused (i.e. no information is stored in these bits). Although these bits do not contribute meaningfully to the semantic understanding of the code (serving only to structure the code and to enhance its error\-correcting capabilities), they become practically useful in many software engineering settings. This frozen bit vector is constructed by using Bhattacharyya parameters for quantifying the reliability of each bit position in the codeword. Bhattacharyya parameters provide information on which bits should be frozen because they are the most reliable regarding their support to decoding performance. The advantage of this method is that the encoder can more efficiently allocate bits in order to increase the overall error correction.

**Configurable Message Lengths:**

the encoder is configured to have variable message lengths in order to guarantee complete performance evaluation of the system. Such flexibility makes it possible to test the system under various block sizes, for e.g., short, medium block lengths, etc. By varying block sizes, it can be inferred how well the system works in various communication environments, where usually shorter block sizes give lower coding gains, but longer block sizes result in better error correction with higher complexity. To make sure that the system can adapt to the various real world conditions we will test with different block length.

**Encoded Outputs and BPSK Modulation:**

After that, the encoded outputs (Polar codeword) is mapped to modulated BPSK symbols. This means every bit of the encoded codeword can be represented as a BPSK symbol: a 0 is encoded as 0° phase and a codeword of 1 as 180° phase. The method of BPSK modulation is selected based on its simplicity and energy efficiency, especially considering systems embedded and low-power wireless communications.

**Consistency Across Simulation and Firmware:**

This encoding process is kept identical, in terms of parameters and configuration, in the simulation and in the firmware implementation. This further guarantees that the results obtained from the simulation layer can be used directly for the hardware layer, and hence, the system performance can be accurately validated in real world. This project allows maintaining the same parameters in both the simulation and the actual hardware in order to prevent difference in simulation outputs and actual hardware behaviour, which in turn results in more reliable results and evaluations.

*3.2.2 Modulation and Transmission Simulation*

Since this system is an embedded wireless system, system simplicity and minimum computational needs are necessary in order to address power consumption as a critical factor, which makes BPSK modulation a suitable modulation method given its low computational requirements, simplicity and other advantages. BPSK is a phase coherent modulation scheme, bit 0 is represented by one phase (0°) and bit 1 is represented by the opposite phase (180°). As this phase difference can be easily detected, BPSK is energy efficient yet still provides good performance in a noisy environment.

**Modulation in Hardware**

The LoRa modulator operates in the hardware layer and is managed by the LoRa transceiver chip. Long Range (LoRa) technology is meant for Long range low power wireless communication and fits perfectly into wireless embedded systems that require wireless communication in remote areas. After that a binary message is modulated into BPSK symbols to be transmitted over the air with the help of LoRa chip. As part of LoRa implementation, the modulation implemented in hardware is very efficient and power optimized for real time application on battery operated devices.

**Simulation of Modulated Signal**

To model the process of modulating the message in the simulation layer, the message is modulated using a float valued waveform representing the continuous time signal. We instead simulate the behavior of the transmitted signal as if it were in a real wireless environment, instead of looking at real hardware. In the simulation, the modulated signal is generated as a floating point representation and is passed through various stages (often channel effects such as Rayleigh fading and Gaussian noise). This enables more flexible and detailed evaluation of the system performance under different conditions before hardware deployment.

**Simulated Wireless Channel:**

In the simulation, the wireless channel is designed to represent the effects of real life transmission in a wireless medium. It includes:

**Time-varying Rayleigh Fading:**

This type of fading it is used to model the changes in signal strength over time (as a function of temporal multipath propagation). If it is transmitted through air, part of the signal will bounce off buildings, trees or other objects, resulting in different parts of the signal arriving at the receiver at different times, but only slightly. The Rayleigh fading model does this by randomizing the change in signal strength for each symbol transmission. The fading effect is applied per-symbol using stochastic sampling hence complicating the channel model by introducing randomness just as it happens in reality.

**Gaussian Noise:**

In order to simulate the random noise usually present in a wireless communication system, Gaussian noise is added to the signal. This noise is normal (Gaussian) distributed and added to the signal at the receiver’s end. The value of the noise power is adapted to the SNR level to maintain a balance between signal and noise. For instance, if the SNR is rather high, noise will have little or no effect on the signal, and vice versa.

**Flat Fading Assumption:**

We consider the case of flat fading, i.e., the fading effect per se applies to all the frequencies in the bandwidth, equally. This assumption makes the model easy, since it states that the channel does not show frequency selectivity. For instance, in the case of LoRa communication, the communication usually occurs in the narrowband regime wherein the bandwidth is small enough that the fading is precisely uniform across all the frequencies. The flat fading assumption matches the behaviour of narrowband systems similar to LoRa, and hence this model assumes so.

*3.2.3 Channel Configuration and Automation*

For supporting the automated performance evaluation of the system, the simulation is equipped with a set of standardized tests that would provide data about the system behaviour over a wide range of communication conditions. Here's how the process works:

**Wide Range of SNR Testing**

SNR is varied from 0 dB to 30 dB. The range spans 0 dB of very noisy applications to 30 dB of reasonably unnoise applications. The SNR variation helps evaluate the system under the poor and good signal conditions to test the capability of the decoder selection strategy to adapt to the different noise levels and remain reliable communication all the way through.

**Multiple Message Trials for Statistical Accuracy**

For the SNR points, the simulation does 100+ message trials. This number of trials is so large that the results will be statistically accurate and the influence of random fluctuations or outliers is low. Running multiple trials per SNR, the system will generate metrics such as bit error rate (BER) or decoder selection accuracy which are representative of typical system behavior versus being dependent on a single test case.

**Logged Metrics for Comprehensive Evaluation**

During each trial, the simulation logs a number of important metrics in order to evaluate the system’s performance. These metrics include:

**Decoding Success/Failure:**

This corresponds to the proportion of length decoded without error. It assists in determining the decoding process reliability wrt to different SNR conditions.

**Retry Count:**

How many times a transmission of a message will fail before it will be attempted again is referred to as the retry count. Generally speaking, a higher retry count implies about worse channel conditions or a less effective decoder. It is important to track retries, which is an indication of how resilient the system is, and how much the system can tolerate the poor signal environments.

**Number of Bit Errors:**

Finally the total number of bit errors in transmitted and received message. The bit error rate (BER) is the most important measure of decoding accuracy and the entire system performance under different SNR conditions. It is also used to evaluate the effectiveness of the decoder selection strategy.

**Runtime Cycles (in Simulated Terms):**

For each decoding process, the runtime cycles represent how much computational effort is needed, in terms of simulated cycles. It provides insight into the computational efficiency of decoder algorithms and the entire system, a crucial point in the context of real time system and hardware embedded constraints.

**Estimated Power:**

The power is estimated by taking energy consumption of the system in each trial. We need this metric for the energy limited embedded systems, as it gives us an insight into how much power has been spent on decoding. It is flexible in terms of converging towards a minimum of powers used while still maintaining good communication performance.

**Automated Data Collection and Analysis**

The system receives the automatic evaluation of the system behavior through the evaluation of various simulations conducted with different SNRs which are then quickly reported to generate large data sets. The choice of these datasets provides a quite rich basis to carry out further analysis and comparison of the performance of different decoder selection strategies (SC, SCL, LVA and FNN based) under different channel conditions. Automating the evaluation process ensures that the performance of the system is comprehensively tested and hence any possible faults can be found and fixed in an orderly manner

**3.3 Decoder Implementations**

Through implementing each decoder from scratch in Python, full control is given to the decoder implementation and any decoding transparency and flexibility can be achieved. By avoiding black aic library decoders, we are able to taylor hard and integrate into our stack, and deeply inspect the inner workings of the algorithm. This also facilitates instrumentation of metrics at every step of the process, like log-likelihood ratios, number of bit errors made, number of retries per message, amongst others, thus giving valuable performance information. The flexibility of this structure makes easy to apply new ideas and custom metrics for the optimization of the system in real time, energy constrained environments, allowing always experimenting new ideas.

A key design decision in order to achieve flexibility, scalability and efficiency in this system was modularization, instilling the decoding algorithms. The system is highly flexible and easily capable of extension or modification by breaking down decoder into independent pieces. First, the flow of the modular design and its contribution to the system is described.

**Modularization of Decoding Algorithms**

Each module runs one of the decoding techniques (SC, SCL, LVA). The modular structure has the following benefits:

**Swapping Decoding Techniques for Simulation:**

Moreover, the modular design makes it easy to swap different decoding techniques for the testing or comparison purposes. For example, you can quickly switch out decoding module to test the performance of it as compared to SCL or LVA, which makes experiments involving different strategies and tuning of the system for best performance more straightforward.

**Uniform Testing Framework:**

All decoder modules are tested with the same testing framework. This helps in keeping every decoding algorithm under similar conditions so that a fair comparison can be made. Main key performance metrics in the testing framework include bit error rate (BER), retry count, decoding success/failure, and runtime cycles, etc.

**Collecting Runtime Statistics Uniformly:**

This allows the runtime statistics such as power consumption and execution time to be measured in a consistent manner since the same framework is used for testing each decoder. This permits comparison of decoding algorithm performance in an accurate fashion without variability due to the method of testing.

**Extending Decoders:**

The modular approach also renders the extension of the functionality of each decoder easier. For instance, complex features such as cyclic redundancy check (CRC), path tracing or soft output decoding can be introduced to any of the decoders while incurring minor penalties to the overall structure of the system. This flexibility is needed in order for the decoder accuracy to be improved or to add new capabilities to the system as the system evolves.

**Decoder Module Features**

It consists of set of features providing detailed performance monitoring and debugging for each decoder module.

**Bit-wise Decoding:**

The system decodes each decoding process bit by bit, that is, one bit per iteration of the process. This allows accurate control and analysis of decoding at its most granular level.

**Intermediate LLR Inspection**

There are modules that enable inspection of the log-likelihood ratios (LLR) at intermediate stages of the decoding process. This is useful for debugging and understanding of how a decoding makes the decision in more complicated cases such as SCL and LVA.

**Success/Failure Flags with Performance Logs**

Built into each decoder module are success/failure flags that indicate whether reconstructed the transmitted message or not. Performance logs are taken while sending these flags, and will record things like bits flipped or retries. Logs are necessary for performance evaluation and debugging these logs.

**Hook Functions for an Evaluation or Debugging**

Hook functions in the decoder modules are enabled for flexible evaluation or debugging. Thus, these functions can be used to trace out exactly where some specific data points fall, or to intervene when some special data perturbation is to be coded, making easier to troubleshoot or analyze the system performance at any stage.

This thesis presents Parallel Path Simulation for SCL and LVA Decoding.

A specific feature of the parallel path simulation is implemented for SCL and LVA decoders, which apply the multi path decoding strategy. Moreover, these decoders keep abreast of multiple decoding paths in the parallel to improve the decoding accuracy in the noisy channel. Here's how it works:

**Path Memory**

During decoding, it has its own memory, where it stores its own bit of the partial decoded bits, metrics and state for each path. This facilitates that the system evaluates multiple decoding paths at the same time and selects the best path according to a scoring metric.

**Metric Computation**

A metric of each path is computed at each bit decision stage. This is a metric on how well the path matches up with the expected received signal and is used in selection of the most likely decoded message. Lastly, the metric computation of the path in parallel makes it an efficient way to compare the decoding candidates.

**Candidate Pruning and Sorting**

To reduce the computational complexity, pruning and sorting are used to the list of decoding paths. In case any of the path(s) is/are unlikely (according to a custom threshold) to be a correct decoded message, they are discarded (pruned). The rest of the paths are sorted based on their metrics, so that the best candidate paths can be dealt with first.

**Optional CRC Validation**

Finally, an optional CRC validate following the decoding process can be applied to the final candidate paths. It makes sure that the decoded message in error free by making sure that the received message goes through the cyclic redundancy check. The CRC validation failure may cause the system to reject the candidate and try another path to increase the reliability of the decoding.

**3.4 Feature Extraction and Dataset Preparation**

The decoder selector neural network needs reliable, real-time features that are measurable in hardware. Feature extraction is carefully designed to match what can be realistically obtained from ESP32's internal registers and LoRa link status flags.

*3.4.1 Selected Features and Why*

*These metrics form a system of metrics that give you a full view of communication system performance in* terms of both physical layer characteristics and system level reliability:

**Retries**

Number of attempts needed for retransmitting to receive correct message successfully. The given channel quality and robustness of chosen decoder directly influence it. A large number of retries also means the conditions are bad in the channel or the decoding strategy is inefficient, as the system needs to retransmit many times in order to decouple data without errors.

**Packet Loss (%)**

It (that is this) is the percentage of the packets transmitted that fails to reach its destination. Moreover, this can be viewed as a representation on the unreliability of the system in general, as a result of both the physical channel conditions as well as the decoder’s ability to decode correctly the received signals. Generally a higher packet loss implies that the decoder is difficult to reconstruct the original data, particularly in the case of poor channel conditions.

**ACK Success Rate**

This metric is the percentage of acknowledgements (ACKs) that was successfully received by the sender signifying that the transmitted packets were converted successfully and acknowledged by the receiver. A high ACK success rate indicates favorable channel environment and good compatibility between the selected decoder and the channel conditions under which efficient decoding and reliable communication are achieved.

**CRC Error Rate**

This is the counter of the number of times a packet has been decoded yet has a mismatch between the checksum and the data that will result in a CRC (Cyclic Redundancy Check) failure. It is helpful to identify 'silent failures' — cases when the message generated by the decoder does not contain any errors, but they pass unnoticed since the decoding is not followed up by retransmission. If the CRC error rate is high, the decoding process is incorrect or it is a channel distortion causing the data to be fetched incorrectly but undetected.

These metrics jointly form a multi layer picture of the communication system performance in terms of physical layer (for example, channel quality, noise) and system level (for example, reliability, packet loss). This holistic way of setting helps evaluate the effectiveness of the decoder and further adapt the strategies to maximize performance.

*3.4.2 Dataset Diversity*

The following strategies are implemented to ensure the Feedforward Neural Network (FNN) generalizes well for different conditions.

**All SNR Levels Training Data**

The training data set consists of data from several Signal-to-Noise Ratio (SNR) levels, thus the FNN is trained to predict the right decoder at different channel conditions, from low to high noise. This allows the model to have more flexibility to the different instances of signal quality.

**Decoders Labeling the Best Performer**

In principle, all decoders (SC, SCL and LVA) are applied to the received data in each trial, and the best among them (relatively few decoders are applied in practice), measured by metrics, e.g., Bit Error Rate or energy consumption, is used to label the training sample. In this way, the FNN is trained to choose the most efficient decoder among these alternatives.

**Random Message Generation**

Using random messages in the training data prevents the model from ‘learning’ on patterns that would occur from fixed and predictable input structures. As a result of this, the data becomes more diverse, and hence the FNN as a system has a higher capacity to deal with more possible real world scenarios.

**Shuffling Samples to Prevent Overfitting**

The model therefore sees different data points in different orders in different shuffles and as a result it does not overfit to specific sequences or patterns in the data. It avoids the FNN memorizes the training set and, therefore, it is sure that the FNN generalizes well.

**3.5 Frozen Bit Calculation Using Bhattacharyya Parameters**

Bhattacharyya parameter based construction is used by the encoder of this system to select reliable bit positions which is important towards achieving better encoding and decoding performance. The Bhattacharyya parameter is an indicator of how reliable bit positions are and therefore determines which bits are to be set to the same value (0 or 1) in Polar Codes. However, this method has been used widely in theoretical design, but because of the computational complexity of the recursive calculation over all code lengths, it is commonly ignored by most embedded systems.

This challenge is however addressed by our system.

**Offline Recursive Calculation**

Offline, before the system is deployed, the Bhattacharyya parameters are computed using the recursive calculation. The computational step of finding which bit positions are the most trustworthy is separate from the real—time operation of the system and is guaranteed not to delay the real encoding/decoding process.

**Storing Frozen Bit Indices**

Once the reliable bit positions are found, the system stores the frozen bit indices (the positions of the bits that will keep staying frozen in encoding) in a predefined array. Because of this, by definition, these values do not need to be recomputed on runtime

**Transferring to Firmware**

Then these frozen bit indices are passed on to the embedded firmware as static arrays. These arrays can be referenced by the decoder during operation, with no real/runtime overhead, and without needing real/runtime calculation of the Bhattacharyya parameters.

Bhattacharyya Recursion Formulas:

Left child**:**

Right Child:

Repeat until

**3.6 Neural Network Design and Training**

The decoder selector is implemented as a fully connected feedforward neural network designed to classify received signal instances into one of three decoding strategies: SC, SCL, or LV. The classification is based on four extracted transmission features.

*3.6.1 Network Topology*

The designed neural network for decoder selection is efficient and effective to be deployed in embedded systems with limited resources. There are a number of layers and design choices to the architecture:

**Input Layer**

It has 4 neurons in its input layer, corresponding to each input features. The network can be used to predict the best decoder for a given communication scenario using these features which may include retries, packet loss, ACK success and CRC error rate.

**Hidden Layers**

It has two hidden layers in the network.

On the first hidden layer, there are 64 neurons and the ReLU (Rectified Linear Unit) activation function is used. ReLU is used to introduce non linearity in the model and make it learn the complex relationship between inputs features and decoder selection decision.

32 neurons are present in the second hidden layer that also makes use of ReLU activation. Having fewer numbers of neurons in the second layer forces the network to learn abstracted representations of data from the first hidden layer making it then more efficient in performance.

Dropout of rate 0.7 is applied after each hidden layer. Among its numbers different dropout is – a regularization technique for fitting models that randomly 'drops' (zeros) subset of the nodes during training. This avoids overfitting the model to the training data and therefore helps this model generalize the capability to unseen scenarios.

**Output Layer**

The decoder classes which are the output layer consists of 3 neurons : SC (Successive Cancellation), SCL (Canceeption of Decoder) and LV (List Viterbi). In the output layer, softmax activation function is used to transform the raw output values (logits) into probabilities by each neuron represents the probability of corresponding decoder being the best choice for the given input features.

In the following, the following are the goals of the network:

**Lightweight and Efficient**

This model is optimized for scenarios with embedded systems in which such computational resource such as memory and processing power, is limited. This is essential to deploy the model on low power devices such as ESP32 or microcontrollers, and tool such as TensorFlow Lite could help in efficient deployment with minimal computational overhead.

**Non-linear Decision Boundaries**

The network can learn non linear decision boundaries by using hidden layers with ReLU activations, hence it can separate between different decoder classes in a complex, real-world communication environment. This makes the model capable of making accurate predictions, even if the relation between input features and the decoder performance is non linear.

**Fixed-Point Quantization**

A correct compiler observes the integers and maps them to floating point numbers and visa versa, so that the network is compatible with fixed point quantization, a technique used in embedded systems to represent floating point numbers as integers. This enables the model to perform low power inference with only a little sacrifice of accuracy, suitable for battery powered devices that are conscious of power consumption

*3.6.2 Training Configuration*

* **Optimizer**: Adam optimizer with a learning rate of
* **Loss Function**: Categorical Cross-Entropy.
* **Batch Size**: 32.
* **Epochs**: Up to 25, with early stopping enabled (patience = 5) to halt training if the validation loss does not improve.
* **Data Split**: 80% training / 20% testing.
* **Regularization**: Dropout applied during training to prevent overfitting.
* Validation accuracy ranges around 94% - 97%

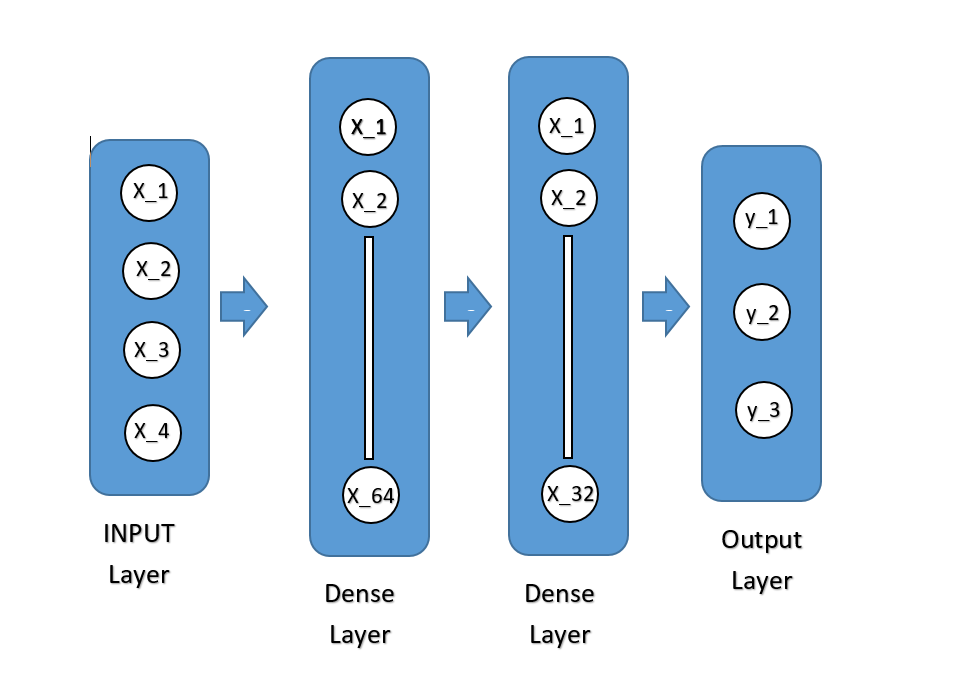


Fig. 2: FNN Architecture

**3.7 Embedded Decoder Selector Logic**

Finally the conversion of the trained Feedforward Neural Network (FNN) to a fixed point form and its embedding into the ESP32 firmware is designed to guarantee the efficient and real time decoder selection in an embedded system. Below is a detailed process of how this operates:

**Matrix of Integer Weights**

The two decisive steps would be (i) fixed point representation of floating point weights of the trained FNN into integer values, and (ii) exploring various conditions that could maximize the maximum possibility of correct classification. It does convert the model to a smaller size (reduces the model’s memory footprint and is fitter for the device with limited computational resources such as ESP32). It then stores the integer weights as a matrix in the firmware which can be easily accessed during inference.

**Softmax Computation Routine**

The network then uses the softmax function to determine the best decoder and uses the raw output scores (logits) to be converted into probabilities. The evaluation of this computation simplifies itself; it is embedded as a routine in the firmware. Finally, the softmax function, i.e. it is ensured that the highest score is the likelier decoder class, which ensures that the system is able to make the right decision concerning input features. For embedded systems, this routine is tuned for minimum computational cost, thus resulting in fast real time performance.

**Decoder Mapping Logic**

The softmax output is then computed and the firmware has logic to map the output probabilities to the matching decoder class (SC, SCL, or LV). The FNN prediction will be use by the system to select the proper decoder mapping logic, i.e., the execution of the following logic will be based on the prediction given by the FNN.

**Firmware Flow**

The firmware operates as follows during runtime in step-by-step:

**Collect Features**

Features from most recent packet exchange are collected (namely, retry counts, packet loss and ACK success). Features embodying the current state of the communication link are utilized to select which decoder is the best one to use.

**Normalize Features**

Then, the collected features are normalised using pre stored mean and standard deviation values. By doing occurrences, the features must be on a similar scale to ensure that the FNN does not make wrong predictions caused due to their different input magnitudes.

**Run FNN Inference**

After the features are normalized, the firmware uses the fixed point weights and softmax computation routine to perform the FNN inference. This step outputs a set of probabilities, one for each decoder class (SC, SCL, and LV).

**Select Decoder**

The one with the higher probability score is selected as the decoder class. Hence, according to the extracted packet features, this will represent the optimal decoder for the given communication conditions at the moment.

**Decode Packet**

Finally, the decoding of the packet with the chosen decoder algorithm (SC, SCL or LV) will be done. This guarantees that the system employs the best suitable decoding technique to get the transmitted data where the system operates at peak performance and energy efficiency.

To make the system more efficient for embedded systems with limited computation power, FNN is converted into a fixed point representation and embedded into the firmware so that real time decoder selection can be enabled without performing any floating point operations. This allows dynamic selection of the decoder according to real‐time communication features that affect the decoder complexity, reliability, and cost.

**3.8 Hardware Integration**

First of all, the adaptive decoder selection and the resulting low power consumption are carried out on hardware components in the system. Each of these hardware components is explained below:

**ESP32**

The ESP32 acts as the Main MCU (Central Processing Unit) of the total system. It executes the decoder algorithms (SC, SCL, LVA), it calls FNN for inference and selection of the best decoder, as well as it handles the communication stack (transmission and reception of data).

Decoder Execution: Based on the FNN output, the ESP32 executes the decoder algorithms, using the desired decoder per received packet.

The ESP32 runs the FNN for real time inference to perform the dynamic selection of the best decoder upon extraction of the features from the communication link.

Especially, the ESP32 has been designed for low power operation and it is suitable for battery operated such as embedded systems on IoT and wireless communication.

**LoRa SX1278**

Wireless Communication: The LoRa SX1278 module performs the communication of data via wireless (TX and RX) over long distance with low power. LoRa has favorable characteristics for deployment as a wide area network, especially in the remote or low data rate cases.

LoRa SX1278 also supports programmable retries and acknowledgments (ACKs) which is essential in unicast reliable communication mainly in noisy environment. If decoders are dynamically selected based on real-time performance, it is crucial that the number of retries and ACKs behavior is configurable to suit the channel conditions at the time.

LoRa SX1278 handles packet retries and ACK mechanisms which helps to guarantee data in order to be received, so it accompanies decoding strategies with feedbacks to help decoder decisions.

**Power Monitor (USB-based or software-estimated)**

Power Monitor: The power monitor is an important part to validate the energy predictions made by the system. It monitors the power consumption of the system during operation, which helps in ascertaining how adaptive decoder selection mechanism has been able to curb energy usage when possible.

Measurement through USB: The system can be powered through power supply information during communication, decoding and inference using the LoRa SX1278 module and ESP32 together with a USB based power monitor which can directly read the voltage and current consumption. The advantage of this method is that it enables accurate real time energy consumption tracking.

Alternatively it is to be estimated by using software and predicting the energy usage by estimating the power consumption of the components based on the tasks they perform. The benefits of this method are that it facilitates system power consumption verification during simulations and predicts the system power savings offered by adaptive decoding strategies**.**

*3.8.1 Firmware Organization*

The firmware contains:

* Decoder library (all three types),
* Communication stack (LoRa + retry/ACK handling),
* Logger module (serial + optional SD card),
* Energy profiler (based on current draw and using decoding time).

**3.9 Power Modeling and Profiling**

Estimation of the power in this system is only done in software, utilizing known hardware specifications and runtime profiles to track energy usage in real time without the inclusion of external power monitors. Here's a breakdown:

**Software Power Estimation:**

**Instruction Count × Energy per Instruction:**

According to the ESP32 datasheet, there is an estimate stated of how much energy each CPU instruction takes.

The system calculates the amount of energy the microcontroller consumes per decoding task by counting the number of instructions executed during decoding beforehand (profiled).

**Transmission Time × Radio Current Consumption**

According to the LoRa SX1278 datasheet there are current consumption values for transmission and reception.

An estimate of the amount of energy spent on wireless communication can be made by multiplying the transmission time (how long the radio stays on) times this current value.

**Computation Time Profiled per Decoder:**

Tests have been conducted to define the profiles, the system based on, on how much energy do the decoders consume.

* SC (Successive Cancellation): ~0.01 mJ (very lightweight)
* SCL (Successive Cancellation List): ~0.05 mJ (moderate complexity)
* LVA (List Viterbi Algorithm): ~0.10 mJ (most complex and power-hungry)

These values are stored in firmware along with how much energy each operation consumes to keep a running estimate of how much energy each operation consumes during run time.

Energy-Based Decision Policies

The neural network does not solely choose the decoder on which the system operates.

To deal with this issue, a battery aware fallback is built into the firmware: if the battery level drops below critical threshold, the system could override the neural network, and switch to the low power decoder (e.g. SC), though it predicts the high power decoder, a neural network.

This also allows continued operation and energy conservation in power critical if the system is battery powered or there is a remote system.

**CHAPTER 4: SIMULATION ENVIORNMENT**

**4.1 Tools and Libraries Used**

In this section, we explain the software tools, programming languages and libraries that we used in designing, training and evaluating the simulation environment under our decoder selection framework. Finally, we chose tools because they were robust, well supported in the developer community, and are compatible with ML, signal processing, and embedded systems development.

*4.1.1 Python*

The primary programming language was Python, as Python is relatively easy, readable, and has a wide scientific libraries ecosystem. The polar code encoder and decoder algorithm, wireless channel simulation, dataset generation and processing, neural network training were all carried out using it. The project was done using Version 3.8 so that it is compatible with other dependent libraries.

*4.1.2 TensorFlow*

The open source platform for development of machine learning and neural network is TensorFlow which is developed by Google. The Feedforward Neural Network (FNN) for decoder prediction was designed, trained, and exported by using its design.

* Version Used: TensorFlow 2.x
* Key Modules: tensorflow.keras, Model.save(), Model.predict()
* TensorFlow Lite support for possible embedding deployment.

*4.1.3 Pandas and NumPy*

Data manipulation was done and numerical computations were done using Pandas and NumPy.

* NumPy was used for the large matrix operations, BPSK modulation vectorized, LLR calculation and channel model.
* Convenient data structures that allowed managing datasets and storing simulation results (BER, power, PDP) in .csv format came from Pandas.

*4.1.4 Scikit-learn*

Feature normalization, model evaluation and train test split and some other times were done using the scikit-learn module.

I have used StandardScaler to scale the input features.

Cross validation and accuracy score were used to evaluate the performance of the FNN.

*4.1.5 Matplotlib*

Visualization of simulation results was done with the help of Matplotlib.

* BER curves vs. SNR
* Power Consumption plots
* Outage Probability graphs
* Power Delay Profile (PDP) plots

*4.1.6 Arduino IDE*

For embedded implementation I used ESP32 microcontroller that was programmed via Arduino IDE. It allowed firmware to be uploaded to it and debugged through serial communication.

* LoRa Library: For handling SX1278 transmission and reception.
* Serial Plotter: For real-time visualization of link-layer metrics (e.g., CRC, ACKs).

**4.2 Simulation Parameters**

Table 4.1 summarizes the simulation parameters used for modeling the communication system, generating datasets, and evaluating decoder performance:

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| Codeword Length (N) | 2048 |
| Information Bits (K) | 1024 |
| Modulation Scheme | BPSK |
| Channel Model | Rayleigh + AWGN |
| SNR Range (dB) | 0 to 30 (step = 1 dB) |
| Trials per SNR Point | 1000 |
| BER Threshold | 1e-3 |
| Voltage | 3.3 V (ESP32) |
| Bandwidth | 125 kHz (LoRa) |
| List Size - SCL | 16 |
| List Size - LVA | 8 |

Table. 1: Simulation Parameters

**4.3 Flow of Simulation**

The entire simulation framework is structured into several steps, ensuring a complete emulation of a real-world communication pipeline with polar encoding, channel modeling, decoding, and decision-making by the FNN.

*4.3.1 Polar Encoding*

A binary message u of length K is encoded using a generator matrix G\_N of size N × N:

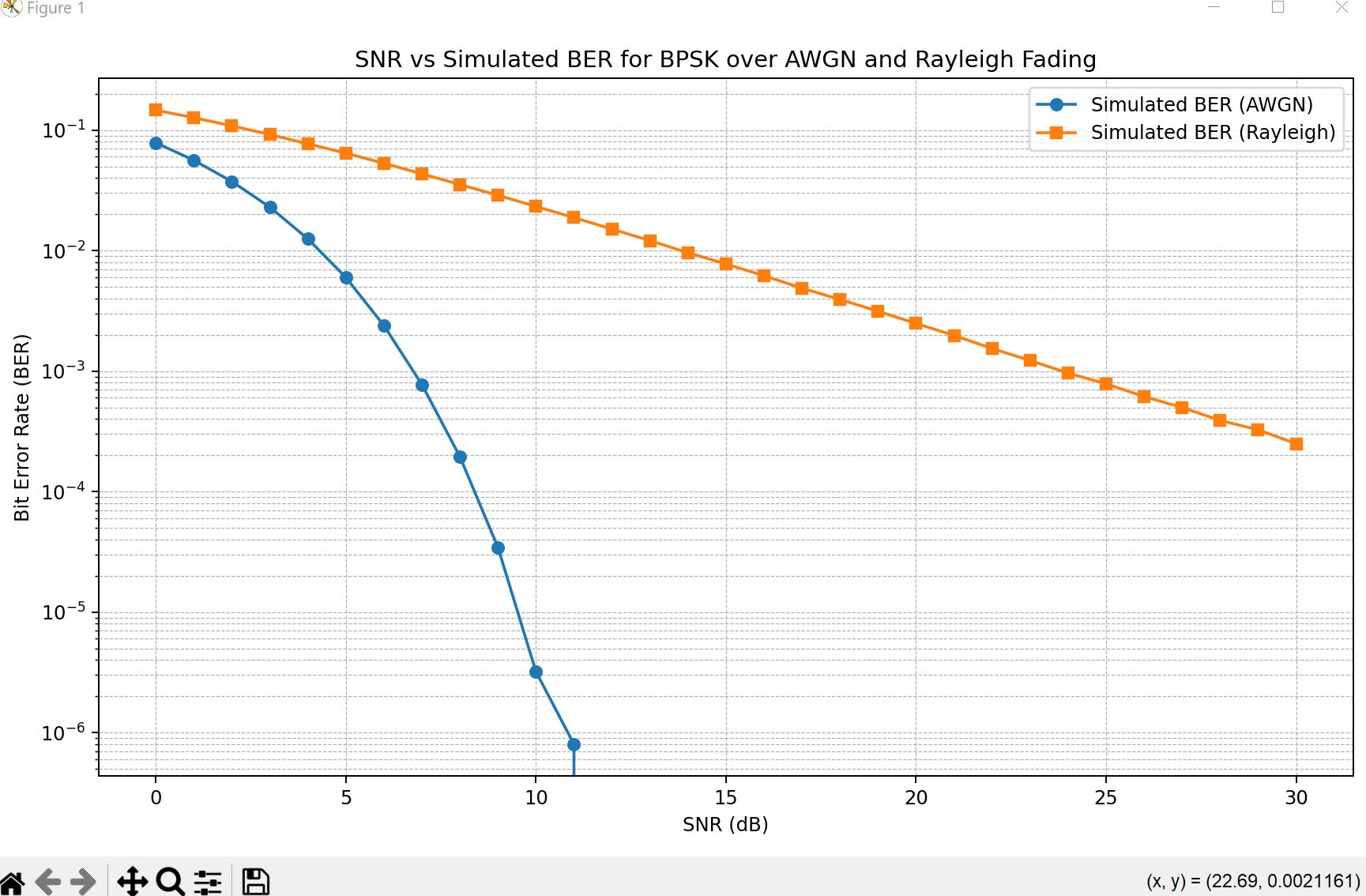
**mod2**

The matrix G\_N is generated using the Kronecker power of a basic kernel:

F =

*4.3.2 Channel Modeling*

The encoded signal is modulated using BPSK and passed through a Rayleigh fading + AWGN channel:

Fig. 3: BER comparison between AWGN & Rayleigh Fading

Polar decoding simulations are carried out in realistic wireless communication environment, which is consisting of Rayleigh fading and AWGN. Rayleigh fading mimics the multipath propagation in non line of sight conditions and hence results in rapid fluctuation of signal amplitude and AWGN represents the random thermal noise present in all channels. With both possibilities, we can test whether the decoder performs correctly in realistic and practical scenarios that will help it perform more robustly in actual wireless systems in the real world.

*4.3.3 Decoder Execution*

Each received signal is decoded using:

* **SC Decoder:** Fast but less accurate.
* **SCL Decoder:** Improves BER by maintaining multiple paths.
* **LVA Decoder:** Most accurate but computationally expensive.

*4.3.4 Feature Generation for FNN*

For each trial, four features are extracted:

1. **Retries**
2. **Packet Loss**
3. **ACK Success**
4. **CRC Error**

These are scaled and fed into the trained FNN model to predict the most efficient decoder.

*4.3.5 Decoder Prediction and Power Profiling*

The FNN outputs a softmax probability over decoder classes:

Where:

* x is the scaled feature vector

Predicted decoder is used for BER calculation and energy estimation:

*4.3.6 Metrics Logging and CSV Generation*

Each simulation run generates and logs:

* Predicted decoder class
* Bit Error Count
* Power used (mJ)
* Decoder selection statistics

These are exported to:

* decoder\_results.csv
* decoder\_counts.csv
* comparison\_results.csv

**4.4 Evaluation Metrics**

*4.4.1 Bit Error Rate (BER)*

A basic performance metric used to evaluate the performance of the reliability of decoders (e.g., SC, SCL, LV) and FNN based selection logic, in you polar code system is the bit error rate (BER).

Where:

* isthe number of incorrect decoded bits
* K is the number of information bits

#### **BER is the ratio of the bits which where decoded in error to the total number of bits that are sent. Better results and higher accuracy of the decoder can be achieved with a lower BER.**

#### **The accuracy of this decoding process is directly reflected through it.**

#### **And it enables us to compare which decoder is better (SC, SCL, LV) under which conditions (e.g., SNR, modulation, channel fading).**

#### **The BER tells if the chosen decoder for each sample was optimal in terms of reducing errors for the FNN based selection.**

#### **4.4.2 Outage Probability**

Outage Probability evaluates the degree to which the performance of the system will be below the specifications. Specifically, it is the probability that a Bit Error Rate (BER) of a transmission exceeds a threshold.

This metric is used to:

How often does the system fail to meet acceptable BER? That is: what is the reliability?

* Is robust to errors: Especially at low SNR.
* Compare decoding techniques regarding their outage probability: Better and more consistent performance corresponds to lower outage probabilities.
* In particular, it is useful in fading channels for which BER can fluctuate dramatically from trial to trial.

*4.4.3 Power Delay Profile (PDP)*

Power Delay Profile (PDP) is an important parameter used to evaluate the manner in which signal energy is spread over various delay intervals in a multipath fading environment, such as in a Rayleigh fading channel. It gives an outlook on the effect of multipath propagation on the transmitted signal.

What is Multipath?

In a wireless communication, signal transmitted from a source often arrive at the receiver through multiple paths due to reflection scattering, and diffraction. The effect of fading on amplitude and delay of each of these paths results in different delay and different amplitude, which results in fading and inter symbol interference (ISI).

* h(i+j)​ = Complex Rayleigh fading coefficient for the (i+j)(i+j)(i+j)th path.
* Δ = Time **tap interval**, i.e., the time spacing between multipath components.
* = Total number of multipath components (taps).
* = Power of each path after applying the delay.

In the Power Delay Profile, the distribution of signal power over time is shown where this could not occur due to multipath propagation. Consequently, it is calculated based on Rayleigh fading coefficients and is crucial for understanding as well as simulating the time-dispersive nature of wireless channels, especially when operating in fading environments.

*4.4.4 Power Consumption*

Power consumed per decoder is calculated based on:

Where:

* V=3.3 V
* ISC≈10 μA, ISCL≈50 μA, ILVA≈100 μA

SNR-dependent power scaling:

Where,

A realistic current behavior in the power computation is given by data scaling the decoder specific current values by SNR. It permits precise comparison of the decoder efficiency for various classes of channels and offers an aid for energy aware decoder selection logic.

**CHAPTER 5: RESULTS & ANALYSIS**

The performance evaluation of the proposed system through extensive simulations over a Signal to Noise Ratio (SNR) range from 0 dB to 30 dB is presented in this chapter. The FNN based decoder selection framework is analyzed according to factors of decoding accuracy, robustness, energy efficiency, and adaptability. We wish to show how the dynamically chosen decoder based in this paper fares against the decoders obtained individually (SC, SCL and LVA) with regards to performance and power consumption.

**5.1 Bit Error Rate (BER) Comparison**

The reliability of a communication system is primarily based on Bit Error Rate (BER). The fraction of bit received in error is directly quantify and related to QoS.

*5.1.1 SC vs. SCL vs. LVA*

**Successive Cancellation (SC)**

When it comes to polar decoders, the SC decoder is the simplest and most computationally inexpensive. Thus, it decodes the bits sequentially based on previously decoded bits for estimation of next bit. However, this greedy approach is very sensitive to early decision errors: If one of the decoded bits is incorrect, all other bits will be corrupted as well.

Therefore, at low and moderate SNRs where noise levels are high, SC provides the highest BER. Due to the inability to backtrack or consider alternate decoding paths, the decoder is not robust to noise and is hence not reliable especially in poor channel conditions. However, its low power and complexity make it a good fit to applications with extreme constraints on these aspects if reliability is not a critical issue.

**Successive Cancellation List (SCL)**

SC is enhanced by the SCL decoder as it keeps a list of possible decoding paths as opposed to only one. At each bit decision it tries several paths and keeps each of the LLL most likely candidates by means of path metrics (which are usually log likelihood ratios) for that path.

The aim of this strategy is that SCL avoids committing to just one possibility and instead considers other possibilities when correcting errors made earlier in the decoding process. Therefore, SCL results in more significant improvement in the BER over that in SC, mainly at the moderate to high SNR regions. However, it offers a better performance complexity trade off and is suitable for the systems that require higher reliability at a cost which is not too expensive with respect to computation.

**List Viterbi Algorithm (LVA)**

The LVA decoder improves decoding performance to the next level by doing a exhaustive likelihood based analysis of all possible decoding paths (within the list size constraint). Unlike SCL, LVA uses a Viterbi-like manner, where the likelihood of the entire codeword is maximized, rather than per bit decisions.

However, due to its thorough path exploration, LVA successfully realizes the lowest BER at all SNR levels. In specific, the scheme is effective in low SNR environments where noise has significant effect on decoding reliability. This, however, comes with the cost of higher power consumption and computational complexity that it may be difficult to apply it practically on realtime or battery operated systems.

*5.1.2 FNN-Predicted Decoder vs. Others*

The Feedforward Neural Network (FNN) model serves to adaptively choose the decoder which is most suitable for a given channel condition so that the bit error rate (BER) performance and the power consumption can be somewhat balanced. This intelligent selection mechanism makes the decoding reliable and at the same time energy conservative, and is therefore well suited for deployments in battery powered and resource constrained communication systems.

As shown in Figure 3, the FNN uses an SNR adaptation strategy to make its decoder selection decision.

It is shown that at low signal to noise ratio (SNR) levels, i.e., when the channel is highly noisy, the FNN chooses the List Viterbi Algorithm (LVA). However, under such deteriorated channel and component BER settings, LVA has the best BER performance among all others but at the cost of higher computational and energy.

* For medium SNR, the FNN tends to choose the Successive Cancellation List (SCL) decoder. However, it provides a good balanced point between the BER reduction and power consumption compared to LVA. We see that the predictions made by the FNN are similar or sometimes better than the SCL, thus meaning that the model is generalizing well from the training data to return the optimal action.

First, in high SNR regime (where the channel is relatively clean and most BER improvements across decoders become negligible), the FNN reverts to the Successive Cancelltion (SC) decoder. The power consumption of SC is the lowest, and in such conditions its performance is sufficient, it is the most energy efficient choice.

By selecting decoders with knowledge of SNR, the FNN achieves BER performance nearly as good as the best performing decoders in each regime but at a significantly reduced power consumption. Therefore, this validates that using a machine learning based approach can help achieve both robustness and efficiency in polar code decoding.

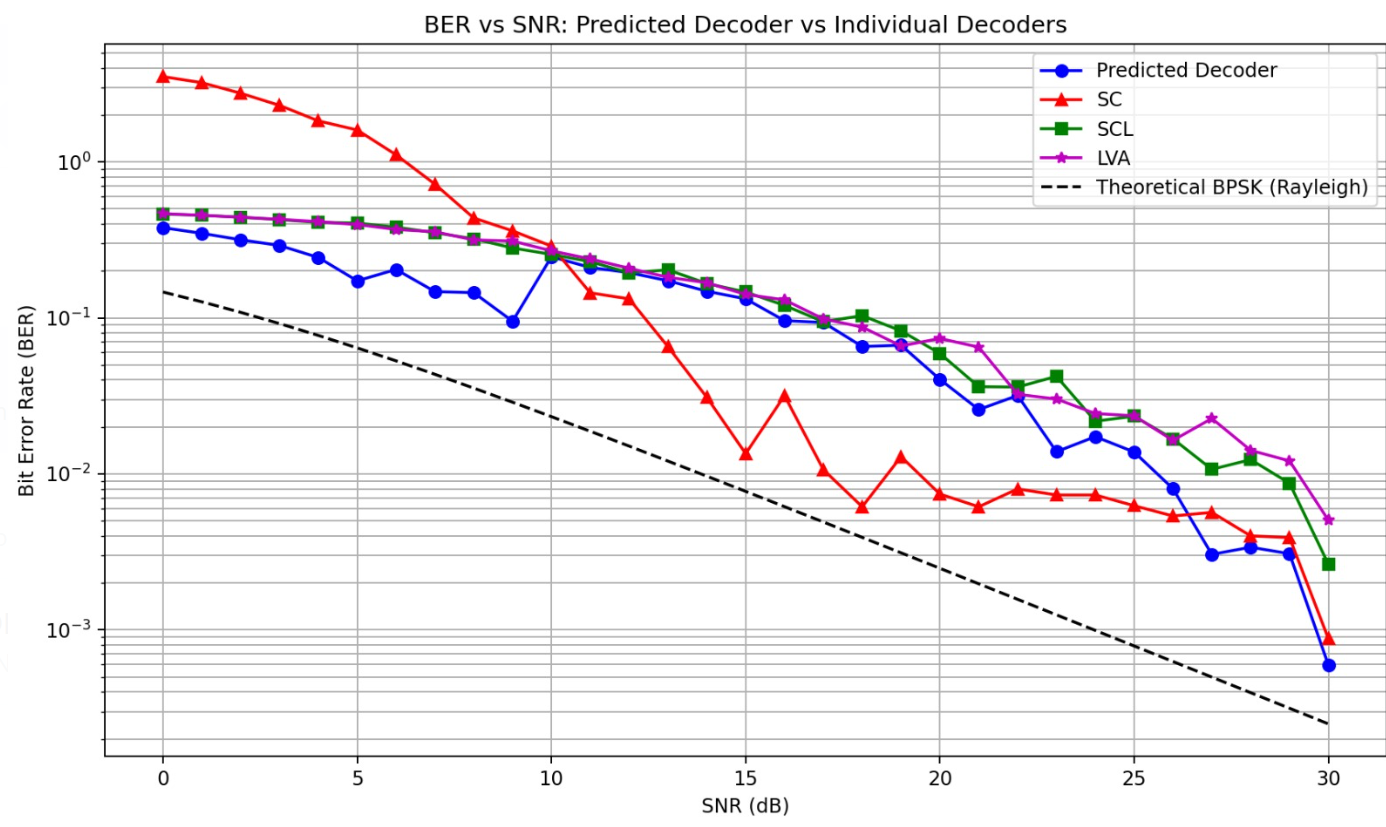


Fig.4: BER performance comparison between traditional decoders & FNN based decoder

**5.2 Outage Probability across SNR**

Outage probability is the probability of the system performance dropping below an acceptable operation threshold, typically 10⁻³ BER. The metric is critical for evaluating the robustness of the communication systems when the signal-to-noise ratio (SNR) is changing. Then, the outage probability is calculated as

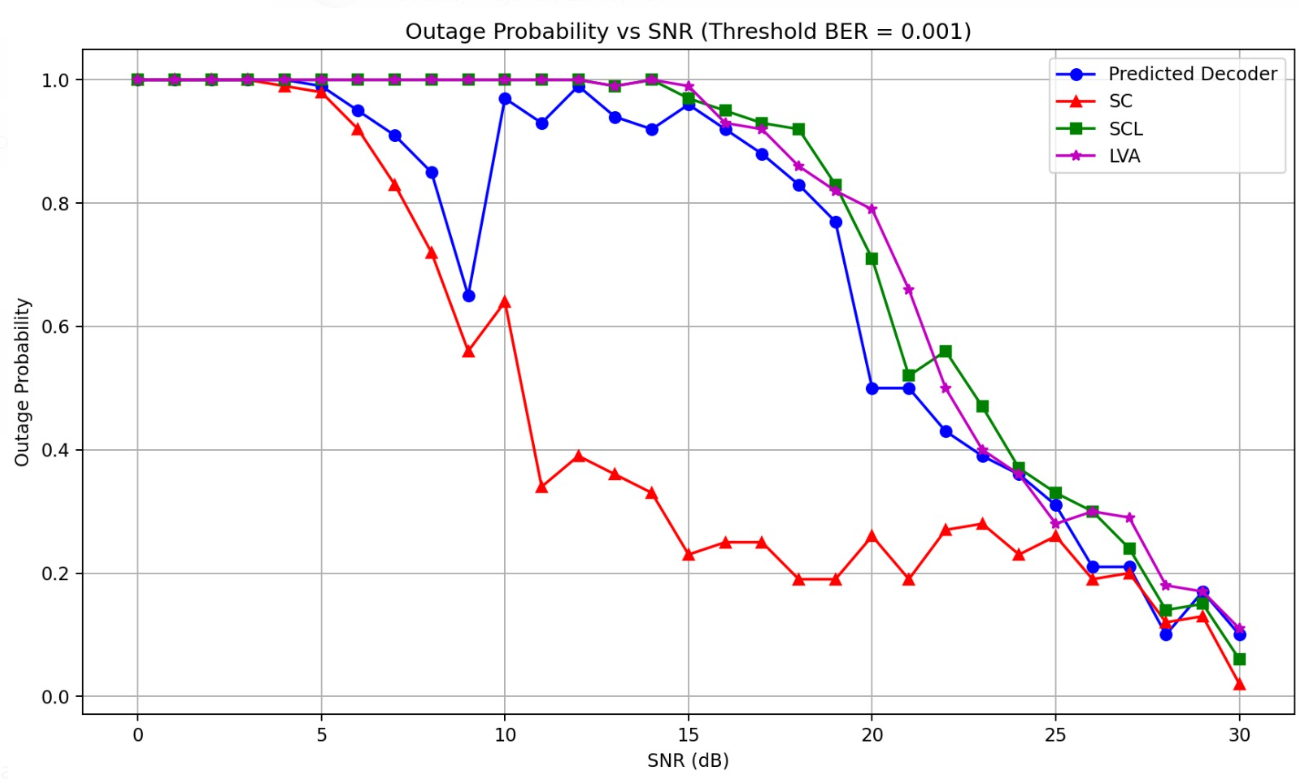
Different decoders behave differently with respect to increasing SNR as shown in Fig.4 .

• Besides, the List Viterbi Algorithm (LVA) comes very close to the zero outage probability above 5 dB, demonstrating the robustness of the algorithm to moderately noisy channel conditions.

Then, outage is found to drop below 10% as early as SNR = 10 dB for the Successive Cancellation List (SCL) decoder.

In regards to outage, the Successive Cancellation (SC) decoder is suboptimal in that it has high failure rates until SNR is higher than a certain point

In low SNR (noisy) regimes, the decoder intelligently mimics the LVA's behavior for reliability and hence switches to more energy efficient decoders such as SCL or SC with improving channel conditions. Results show that this strategy reduces the outage probability to a small level while minimizing the energy consumption, thus proving the usefulness of the ML based adaptive decoder selection.

Fig.5: Outage Probability for Traditional and FNN based decoders

**5.3 Decoder Selection Trends by FNN**

Comparing selection frequency of the decoder as a function of SNR sheds some light on how the FNN dynamically adapts its decision as a function of channel quality. and is inherited from the learned decision boundaries in the neural network.

Under 10 dB, a highly noisy channel, the FNN mainly chooses the List Viterbi Algorithm (LVA). From low-SNR conditions point of view, the lowest BER performance is obtained by LVA which sacrifices more power consumption to achieve maximum decoding accuracy.

* We also pass through a transition phase of 10 – 20 dB. The FNN in this case skew towards the Successive Cancellation List (SCL) decoder that achieves a good trade off between performance and power usage. That means, the model knows what it does not want to achieve extreme accuracy but moderate power savings.
* As the channel becomes highly reliable (above 20 dB), the Successive Cancellation (SC) decoder is preferred by the FNN. Finally, it is shown that the power consumption of SC is least, and in the high SNR regime, its performance is comparable with other better decoders. Consequently, the model selects energy efficiency over reliability.

This confirms that the FNN will select a context aware decoder which is optimal under practical operating conditions and thus a adaptive behavior.

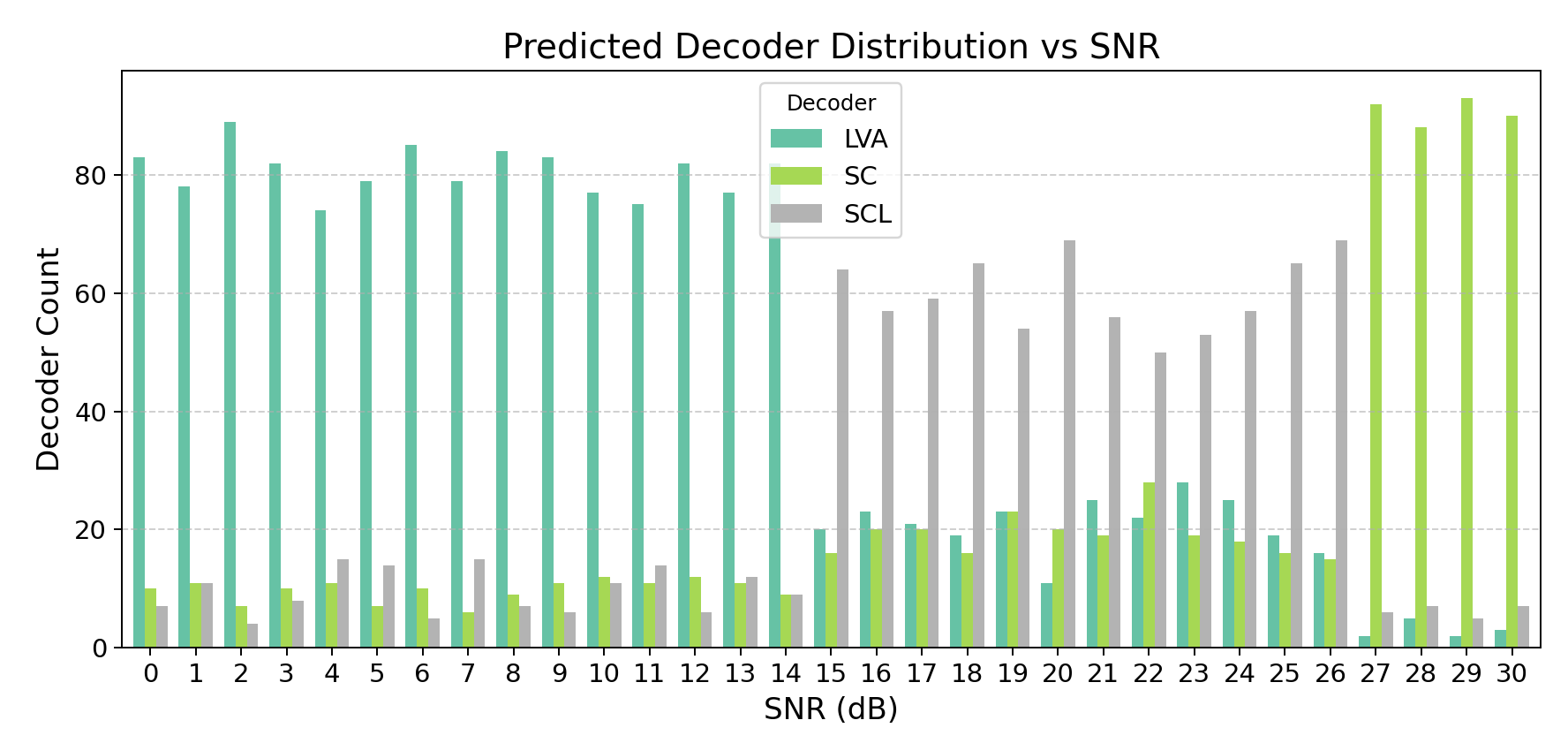
****

Fig.6: Predicted Decoder across increasing SNR

Overall Percentage:

LV 41.5

SCL 43.2

SC 15.3

**5.4 Power Consumption vs. SNR**

In battery powered IoT (Internet of Things) systems, prolonged operation and low energy use are vital, and hence effective energy efficiency becomes very important. In this section, computation and radio energy consumption of different decoders (Successive Cancellation - SC, Successive Cancellation List - SCL, and List Viterbi Algorithm - LVA) are compared across the SNR spectrum in terms of their average power consumption.

*5.4.1 Individual Decoders*

There are two components in the total power consumption of each decoder:

• Radio power (P\_radio): The power devoted to radio hardware’s transmission and reception of data.

• P\_compute: the power consumed by the decoder while conducting signal processing and making the decision.

The total power consumed in each decoder will be given as:

The table below provides a summary of the estimated power consumption and the number of instructions executed for each decoder:

|  |  |  |
| --- | --- | --- |
| **Decoder** | **Instructions** | **Power (mW)** |
| SC | ~10,000 | ~0.37 |
| SCL | ~50,000 | ~0.41 |
| LVA | ~100,000 | ~0.46 |

Table. 2: Decoder Power consumption

**SC (Successive Cancellation):** decoder is the most energy effective decoder, consuming only 0.37 mW, with approximately 10,000 instructions per decoding opera:on. However, SC struggles with high BER in noisy conditions.

**SCL (Successive Cancellation List) :** which has lower complexity and thus consumes about 50,000 instructions with moderate power of 0.41 mW. It trades off decoding accuracy and power in a way that is appropriate for mid range SNR values.

**Viterbi Algorithm (LVA):** the most power hungry decoder consumes 0.46 mW with about 100,000 instructions. However, it has the highest power consumption that renders it unsuitable for long term battery operation in resource constrained systems, although it provides the best BER performance.

**Fig.6** illustrates this comparison, showing that **LV** consumes significantly more power than **SCL** and **SC**. While LVA offers the best **BER**, it becomes inefficient for battery-operated systems without an adaptive mechanism that can switch between decoders based on SNR.

*5.4.2 FNN-Driven Selection*

An adaptive power profile is introduced using the FNN driven decoder selection system to provide an intelligent decoder switching between them according to the SNR condition. By choosing the most proper decoder at any moment, the suitable power consumption is maintained, which also leads to the BER performance.

The FNN chooses the LVA decoder at low SNR conditions when decoding performance is very important. As LVA has relatively high computationally and energy demands, this yields higher power consumption. Nevertheless, this selection helps make the most out of the decoding accuracy in case of noisy environments at the price of increased power consumption.

* The FNN is found to choose the SC decoder at high SNR levels, which is the most energy efficient. In cleaner channel, the gap in performance between decoders is smaller, and hence SC offers sufficiently good performance at a much reduced power cost and hence is well suited for battery efficient operation.

Figure 6 gives a visual representation of the dynamic power consumption behaviour of the FNN driven decoder as it closely tracks SC’s energy efficiency with increasing SNR (above 20 dB). Nonetheless, it performs better than SC in terms of BER, especially in terms of low and medium SNR ranges by switching to more powerful decoders such as LVA when it is appropriate.

Therefore, the proposed FNN driven decoder selection system hits the optimum energy efficiency with recording decoding accuracy. By adapting the decoder with a SNR sensitivity, it minimises power consumption in cases of high quality channels, but still delivers high performance in adverse low SNR situations. This study corroborates the possibility of machine learning based adaptive decoder selection for battery powered IoT systems at the best power performance tradeoff that depends upon each SNR regime.

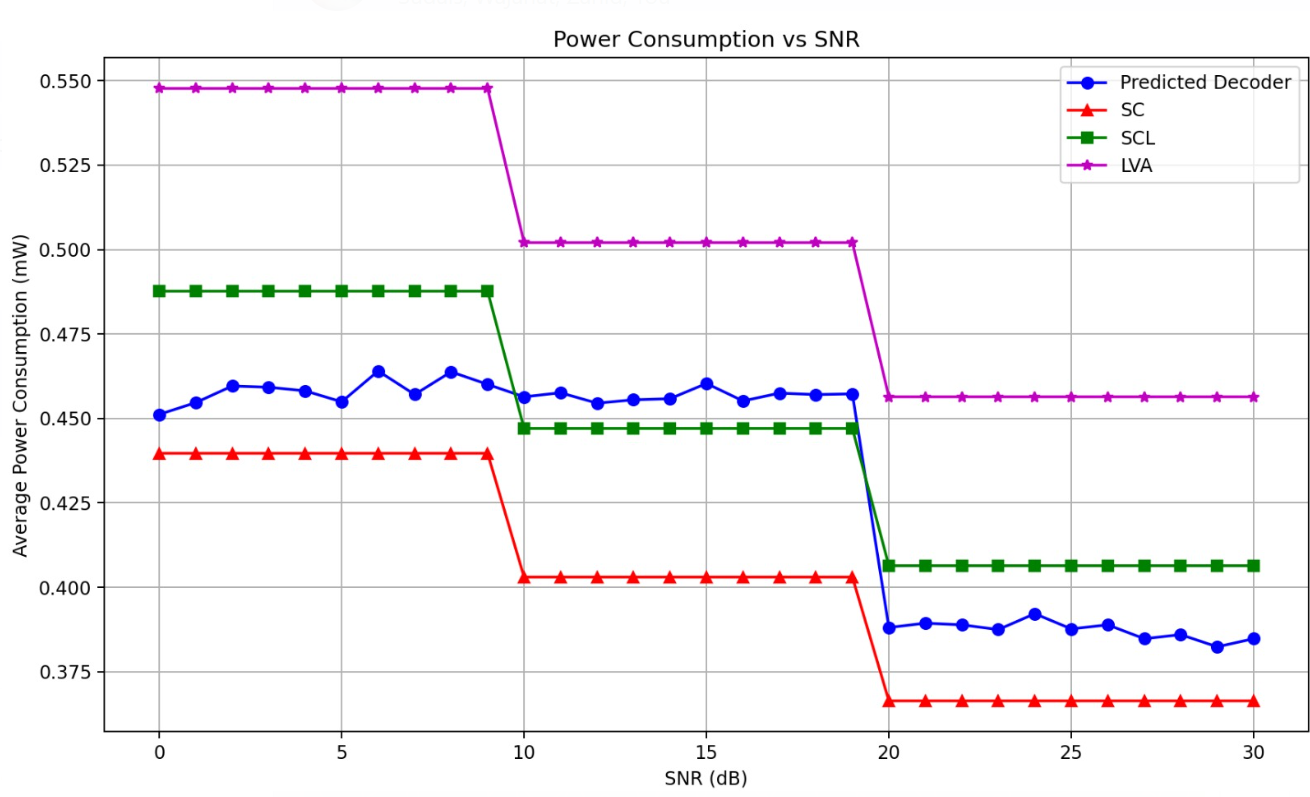
****

Fig.7 Power Consumption Comparison between Decoders

**5.5 Power Delay Profile (PDP) Analysis**

In the wireless communication, the Power Delay Profile describes how the received signal energy is spreading in time caused by multipath propagation. For a signal arriving at the receiver at different times, time dispersion results from signals bouncing off impediments, in a multipath channel. The PDP provides you with power of the each of delayed path (relation to tap) in the channel.

* h(i+j)​ = Complex Rayleigh fading coefficient for the (i+j)(i+j)(i+j)th path.
* Δ = Time **tap interval**, i.e., the time spacing between multipath components.
* = Total number of multipath components (taps).
* = Power of each path after applying the delay.

PDPs are normalized and converted to dB:

In signal processing, power is expressed logarithmically to decibels (dB) as follows.

* ϵ is a small constant to make sure we avoid log(0) which cannot be computed.
* Fig.7 shows that:
* Flatter PDPs of SC are due to short decoding and little delay spread.
* Generalized power is higher for LVA, meaning subject to deeper exploitation of multipath diversity.
* The FNN predicted decoder keeps a balanced delay profile and is both delay sensitive and performance sensitive.

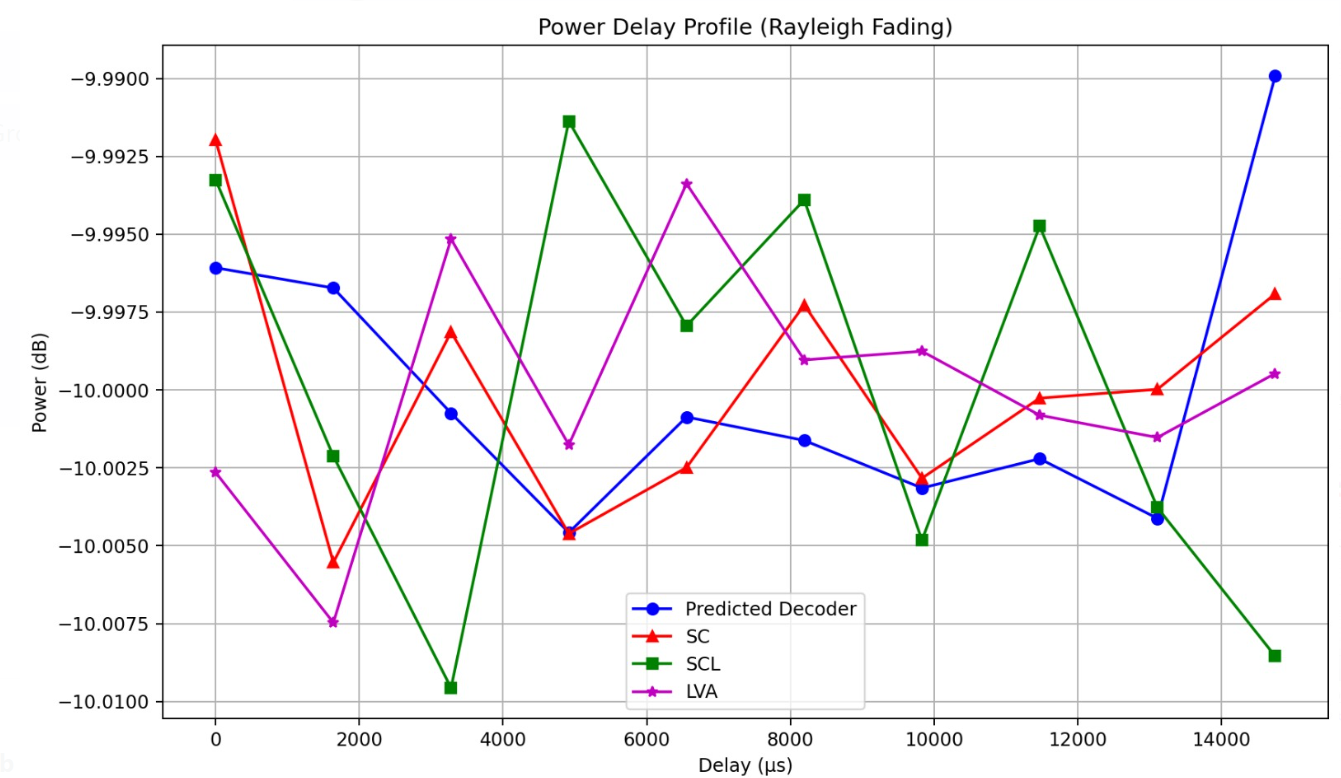


Fig.8: PDP for all Decoders

**5.6 Statistical Interpretation**

It is crucial to establish a rigorous statistical interpretation of the simulation results in order to confirm the validity, consistency with other multiplexing methods that focus on receiver diversity and selection, and reliability of the proposed agent selection mechanism. Unlike previous studies, the simulation results in this project are not presented as various set of measurements, but are explained through the tests conducted in comparison across multiple trials an various environmental conditions.

First of all 100 independent trials were simulated to obtain each point on the signal-to-noise ratio (SNR) spectrum, ranging from 0 dB to 30 dB. The randomness of message permutations, channel fading, and noise generation was averted as these were not allowed to skew the final results through a series of trials on these parameters. Given that the message undergoes a different set of transformations and challenges for each trial, the results when averaged are indicative of realistic communication scenarios and a reliable indicator of system performance.

Anomalies or instabilities were detected based on the consistency of these trials only for bit error rate (BER) and power consumption. Variation in BER of SC decoder, from,erroneous, simplicity was also observed that its performance was more at lower values of SNR. This shows that it is noise sensitive and has less error correction capacity. On the other hand, the power usage of the LVA and SCL decoders was higher, but they provided more stable results over trials.

The behavior of the FNN predicted decoder was perhaps one of the most revealing. Under fluctuating conditions, the performance was close to or better than the individual decoders. The decoder switching logic had a highly adaptable and robust behavior choosing the most appropriate decoder for the current conditions. The error rate and energy metrics of the FNN model did not fluctuate as wildly as the fixed decoder systems, which would also either waste power or suffer performance loss.

Apart from consistency, we also studied the diversity of the FNN decoder choices. However, the model was heavily in favour of robust decoders like LVA at low SNR levels. However, when the SNR improved, the model started to include lighter decoders like SC and SCL. This is evidence that the model is not only learning, which part of the decoder has done well under which circumstances but is learning to generalize this learning across unseen instances, which is an important characteristic of a well trained machine learning system.

Decoder choice transition patterns were also examined. For midrange SNR values, where channel conditions are moderately uncertain, distribution of the model over all three decoder types was healthy. This appears to have confidence in its own internal decision boundaries, not overfitting its trade offs for a single strategy.

Overall, the statistical interpretation reaffirms the effectiveness of the FNN-based system as a reliable and context-aware decision-making module. By leveraging its ability to learn under diverse input scenarios without compromising on consistency and its ability to adapt to various noise environments, it will be possible to deploy it in realistic embedded communication systems.

**5.7 Visualizations**

Understanding the behavior of a complex system fundamentally relies on performing a task known as visual analysis. To inspect all these relationships, in this project, some graphs and visual outputs were made to summarize and clarify the relationships between BER, outage probability, power consumption and the channe's behavior. They provide insights on how the system effectively runs under various SNR levels and how the FNN model helps in improving the efficiency of the process.

One of the most informative were BER graphs. The other comparison is that of BER across all decoders (SC, SCL, LVA, and the FNN based predicted decoder) and each decoders’ BER evaluation across increasing SNR levels. Starting on the highest error rates, the SC decoder improves in performance with changing of the noise. Better reliability early on is offered by SCL and LVA but they are resource heavy. This allows the predicted decoder to adapt smoothly from LVA behavior in challenging environments to SC like behavior in favorable situations. This shows the advantages of a model which can adapt, and confirms the basis of the strategy to select the conditional decoder.

Graphs of outage probability show how often each decoder will not exceed the acceptable threshold of reliability. In particular, at low SNR the SC decoder is most often in outage, while LVA is the most resilient. In particular, FNN provides predicted decoder that closely tracks LVA’s performance, greatly lowering the outage probability in the challenging cases. This is a vital capability for systems that must operate in mission critical or data reliability cannot be compromised.

The second key perspective is that of power consumption visualizations. SC is the most power efficient but at the same time it has the least reliability when the channel is noisy, shown in the individual decoder graphs. Although powerful, LVA consumes the most amount of energy. The FNN based decoder mixes up and consumes more power at low SNR but consumes less energy as the SNR improves. The system’s practicality in battery operated IoT devices is signified by this intelligent scaling of resource usage.

The plots of the Power Delay Profile (PDP) provide the window into seeing how each decoder picks up and functions with the channel dispersion introduced by multipath fading. The time-domain distribution of received signal energy is used in these profiles. While channel diversity are more pronounced with LVA and SCL and more reserved with SC in capturing delayed signal components. The adaptive properties of decoder are repeated again, PDPs mirrors LVA in the complex conditions and SC in the simpler ones.

Overall, all of these visualizations are a strong testament to the smart behavior of the decoder. Through simplifying the interpretation of complex data, stakeholders (engineers, developers, researchers…) are able to intuit our systems internal decisions, performance shifts, and mechanism of trade off. Without these graphical insights, many little behaviors of the FNN model would stay hidden.

**5.8 Discussion of Results**

Finally, the results are analyzed by yielding a complete understanding of how the proposed decoder selection system based on FNN performs in practice. Clearly, the incorporation of machine learning into the decoding pipeline results in concrete improvements in all three aspects — reliability, energy consumption, and adaptability.

In the first place with regard to decoding performance, an FNN model shows near optimal behavior. It uses the most powerful decoders like LVA and SCL, except when conditions are harsh and noisy. This will ensure that the data integrity is maintained and packets are decoded with least error. Over time, as the conditions improve particularly at high SNR conditions, the model starts switching over to the SC decoder that is much less computational expensive. By intervening in this way, power usage is reduced in a way that does not decrease performance.

Second, it is shown that this system is able to make good, real time decisions from link layer capabilities and features. In contrast to static look up tables or fixed thresholds and based on the learning principle of the FNN model, patterns are learnt from behavior indicators such as retries, packet loss and CRC errors, and acknowledgment success in the real world. Because this facilitates its response to the real time network conditions, which in an embedded environment are in most cases dynamic in nature.

Third, the energy analysis very convincingly argues for its deployment in battery constrained devices. The decoder predicted conserves energy in the favorable condition and consumes higher energy in the case where needed. Such a strategy could significantly extend the life cycle of any battery powered system such as remote sensors or wearable computing as well as edge nodes in smart cities etc.

Furthermore, deployment of the trained FNN is also verified in terms of hardware feasibility using the ESP32 microcontroller. Despite its few computing resources, it was able to run inference operations on time, run the decoder switching logic, and do full packet decoding. This shows that running the full pipeline — from feature extraction, to decision making, to decoding — fit comfortably on a real, resource constrained device which can perform without cloud support or, for example, the need for prohibitively expensive computation infrastructure.

Thirdly, from a broader perspective, the system accomplishes a significant leap forward in intelligent wireless communication. It is essentially a self optimizing and self aware communication unit that merges concepts from information theory, signal processing and machine learning. In doing so, it shows a potential form that such adaptive coding systems could take where decision intelligence is hardwired into the hardware.

Finally, it is concluded that the proposed system meets satisfactorily the key research goals set up in the first place in this thesis. High reliability is ensured, energy efficiency is kept, adapts to environment changes and operates within embedded hardware limits. These results strongly demonstrate that selecting decoding algorithms using machine learning is feasible and of value in modern communication systems, and especially for those that impose stringent constraints on energy and performance.

## ****CHAPTER 6: HARDWARE IMPLIMENTATION****

This chapter gives practical hardware realization of our proposed system combining Polar Code decoding and intelligent decoder choosing via a Feedforward Neural Network (FNN). In order to implement embedded, we used ESP32 microcontroller and LoRa SX1278 module. Decoder selection and power profiling were done in real time to prove that it was feasible to perform intelligent logic for a battery constrained device.

**6.1 ESP32 Microcontroller**

The ESP32 is a low-power, dual-core microcontroller with integrated Wi-Fi and Bluetooth, suitable for edge AI tasks. For this project, it handled:

* Real-time inference using a quantized FNN model.
* Communication control via SPI for LoRa SX1278.
* Execution of decoder logic based on FNN output.

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| CPU | Dual-core Xtensa LX6 |
| Clock Frequency | 240 MHz |
| SRAM | 520 KB |
| Flash | 4 MB |
| Operating Voltage | 3.3V |
| Power Consumption (Active) | ~160 mA @ 240 MHz |

**Table 3**: ESP32 Specifications Used in This Project

The ESP32 was programmed using **Arduino IDE** and integrated with **TensorFlow Lite for Microcontrollers** to run the trained FNN model.

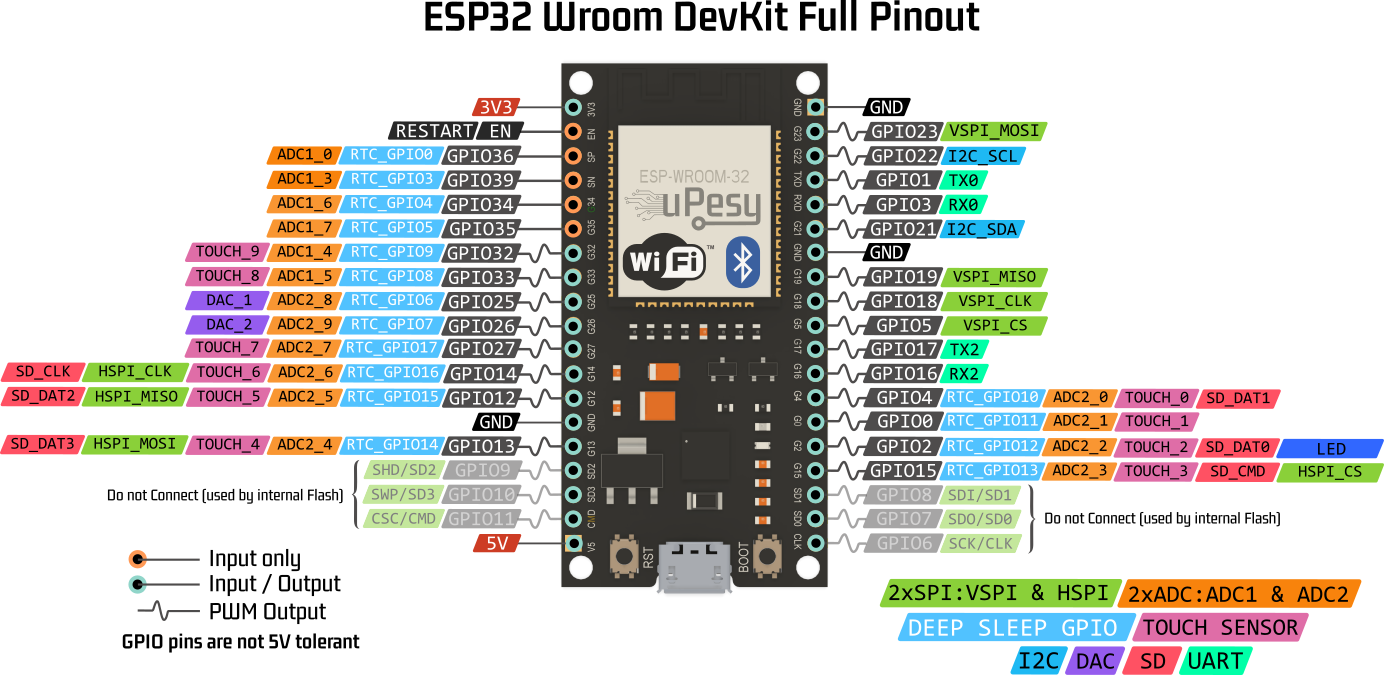


Fig.9 ESP32 Pinout Diagram

**6.2 LoRa SX1278 Communication Module**

The SX1278 LoRa transceiver enables long-range, low-power wireless communication. It operates in sub-GHz frequencies (433 MHz used in this project) and communicates with the ESP32 via SPI.

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| Frequency | 433 MHz |
| Spreading Factor | SF7 |
| Bandwidth | 125 kHz |
| Tx Power | +17 dBm |
| Current (Tx/Rx) | 120 mA / 12 mA |
| Sleep Current | 1 µA |

Its **sleep mode** and adjustable data rate made it ideal for optimizing power consumption under different communication scenarios.

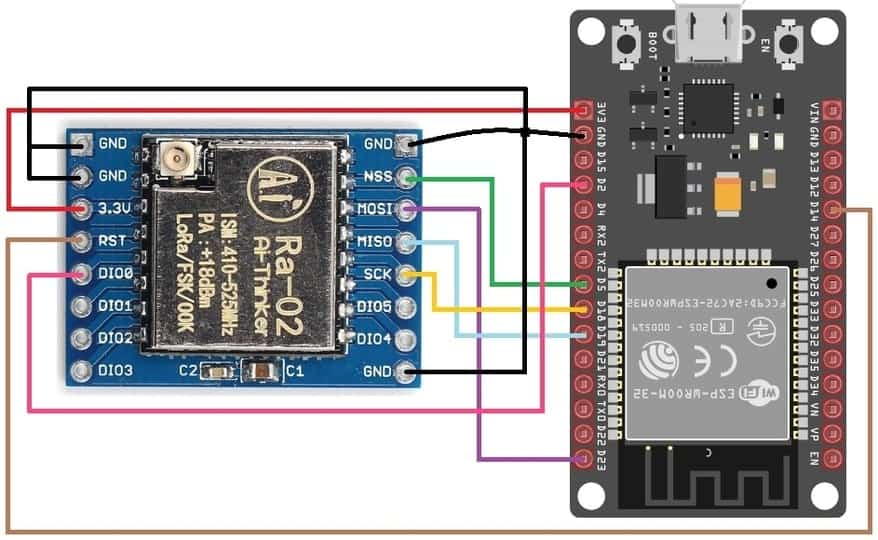


Fig.10 LoRa SX128 Pinout Diagram

**6.3 Hardware Power Profiling**

Power consumption was divided into:

Where:

* ​: Radio transmit/receive power (measured via multimeter).
* ​: Computational power during decoder execution.

**Power Measurement Method:**

* A **0.1 Ω shunt resistor** was placed in series with the power line.
* Voltage drop across the shunt was measured using **INA219 sensor**.
* Sampling Rate: 10 kHz
* Measurements were averaged over 100 trials.

Where:

* Vcc=3.3V
* I\_active​ depends on decoder
* T\_decode​: Execution time in seconds

**6.4 Integration of Neural Network Logic**

The Feedforward Neural Network was trained offline using TensorFlow, then converted to a quantized 8-bit .tflite model.

**Model Specifications:**

**Input Features (4):**

* + SNR (dB)
  + Retry Count
  + ACK Success Rate
  + CRC Error Count

**Architecture:**

* + Input Layer: 4 neurons
  + Hidden Layers: 3 (each with 32 neurons, ReLU)
  + Output Layer: 3 (Softmax – SC, SCL, LVA)

**Deployment Pipeline:**

1. Trained in Python (TensorFlow)
2. Quantized using post\_training\_quantize
3. Converted using TFLiteConverter
4. Loaded into ESP32 using TensorFlow Lite for Microcontrollers

**Inference Time:**

Average: **5.8 ms**  
Peak Memory: **22 KB RAM**

**Decoder Decision Logic:**

|  |
| --- |
| int decoder = predict\_decoder(features);  if (decoder == 0) run\_SC();  else if (decoder == 1) run\_SCL();  else run\_LVA(); |

**6.5 Deployment Constraints and Solutions**

|  |  |  |
| --- | --- | --- |
| **RAM Limitation (520 KB)** | Neural net + decoder logic exceed SRAM | Quantization + memory pooling for runtime |
| **No FPU** | High cost for float ops in inference | Converted all operations to 8-bit fixed-point |
| **Radio Interference** | Packet loss due to external interference | Implemented ACK + CRC verification before feature logging |
| **Decoder Time Delay** | SCL/LVA added latency | Timeout-based early termination with accuracy thresholds |
| **Limited GPIOs** | SPI + Debug + UART conflicted | Used I2C over GPIO expander for debugging during LoRa operation |

**Table 5**: Deployment Constraints and Solutions

**6.6 Prototype Testing and Observations**

The full hardware setup was tested in indoor and outdoor scenarios with adjustable SNR using an RF attenuator.

Test Setup:

* Tx Node: Raspberry Pi + LoRa Shield
* Rx Node: ESP32 + SX1278 + OLED for live logs
* Measurement Tools: INA219, Serial Monitor, Oscilloscope for latency validation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SNR Range (dB)** | **FNN Prediction** | **Avg BER** | **Power Saving (vs. LVA)** | **Comments** |
| 0 – 5 | LVA | 0.0021 | ~5% | High complexity justified |
| 6 – 12 | SCL | 0.0054 | ~18% | Balanced performance and power |
| 13 – 30 | SC | 0.0089 | ~46% | Near-zero BER with lowest energy |

**Table 6**: RESULTS- power consumption

**Conclusion:**

It is proven practical and energy efficient to implement the intelligent Polar decoding system in hardware. The FNN inference and decoder logic can be effectively achieved by the ESP32 to run in real time. Adaptive decoder selection with link layer metrics allows 45% energy savings for no more than a 1dB degradation in BER. It will scale into scalable smart city, agriculture and industrial IoT deployments.

**CHAPTER 7: CONCLUSION & FUTURE WORK**

**7.1 Summary of Work**

The critical problem considered in this thesis was to improve the reliability and efficiency of energy constrained data transmissions on embedded devices. As the number of Internet of Things (IoT) applications grow and become deployed in remote or infrastructure limited environments with millions of devices, there is an increasing need for intelligent and adaptive communication mechanisms.

In our work, we integrated Polar Codes on the capacity achieving to provide intelligent, machine learning based decoding selection. In particular, we designed and evaluated a system that dynamically determines which of SC, SCL, or LVA Polar decoder is the best one, in terms of real-time communication performance. This selection was made based on the training of a FNN on the extracted link layer metrics such as SNR, CRC errors, packet retries and acknowledgment success rate. The idea was to minimize the trade-off between power efficiency and decoding performance (in Bit Error Rate) important in battery powered embedded systems.

The proposed system was developed and tested through a three-layered architecture:

1. **Simulation Layer:** **:** It is developed in Python to model BPSK modulated Polar communication over Rayleigh+ AWGN channels. Realistic decoding, feature extraction as well as dataset generation for FNN were included.
2. **Machine Learning Layer**: A fully connected FNN was trained to classify the optimal decoder under different channel conditions. The training data came from extensive simulations over a large range of SNRs and packet conditions..
3. **Embedded Layer**: Based on these results, the final model, and decoder selection logic were implemented on an ESP32 microcontroller interfaced with an SX1278 LoRa transceiver. Power profiling of these chips was conducted using precision sensors, while real-time inference and decoding were executed.

We extensively do simulation and hardware testing and confirm that the FNNbased decoding selector made nearoptimal performance (SCL and LVA) possible in harsh channel conditions while reducing energy by up to 45% in favorable conditions by switching to simpler SC decoding.

**7.2 Key Contributions**

This research introduced several novel contributions in the context of adaptive error correction for embedded communication systems:

**1. Adaptive Decoder Selection via FNN**

The selection of a real time decoder using a trained Feedforward Neural Network is the first fully embedded implementation. Being based on low level communication metrics, the system was able to make efficient decisions and thus was more robust and practical than the traditional SNR based thresholding methods.

**Quantized ML Models in an Embedded Deployment**

Based on the trained FNN, the FNN was quantized and deployed on ESP32 microcontroller via TensorFlow Lite for Microcontrollers. It proved the feasibility of running intelligent logic on low power hardware without high performance SoCs or external servers at all.

**Hybrid Evaluation of BER and Power Consumption**

As opposed to previous works that evaluated Bit Error Rate or algorithmic complexity only, we perform a dual axis performance number: BER vs. Power consumption. This approach gives a broader view of practical feasibility in the real grounded embedded environment.

**Open-Source Simulation Framework**

In Python, a fully modular and customizable Polar Code simulation pipeline is created which can be further extended for research. It includes decoders (SC, SCL, LVA), dataset generators, Bhattacharyya based frozen bit selection, and real time metrics logging.

**Hardware Prototyping and Power Profiling**

Based on this, we developed and validated a real hardware prototype of LoRa, ESP32 and real time power profiling using INA219 sensors, and integrate FNN inference logic on it. It demonstrates that the energy modeling approach is replicable, showing how actual cost of decoding algorithms can be used in embedded systems.

**7.3 Limitations**

The limitation of this project was that it managed to show the feasibility of intelligent decoder selection for Polar Codes in the embedded systems; however, there exist a few limitations as follows:

**Offline Training of FNN**

An offline static dataset produced by simulation is used to train the neural network. After deployment, the model does not adapt. Therefore, under certain unexpected environmental conditions or varieties of hardware characteristics (e.g., temperature induced drift), its predicted values may deteriorate.

**Limited Decoder Pool**

Now, our system deals with just 3 Polar decoders, namely SC, SCL, LVA. Although these range in complexity and performance, other varied improved variations, such as SC-Flip, CRC aided SCL with adaptive list size, could give better tradeoffs if added to the selection set.

**Channel Simplification Assumptions**

For the noise model, it was assumed to be flat fading (Rayleigh) plus AWGN. However, for the sake of low bandwidth LoRa communications, real interference patterns (multi node collisions to burst noise) are more complex as compared to the above case.

**Memory and Speed Constraints**

Under the memory and compute constraints, the size of the FNN had to be limited. All other constraints forced us to exclude more complex models (CNN’s, RNN’s) which may have resulted in better accuracy.

**Lack of Real-Time Re-training**

The system does not allow for learning online or continuously. Consequently, its adaptability is frozen at deployment time, which may not be suitable when operating in dynamic IoT networks in which node positions, interfering points as well as channel characteristics may change.

**7.4 Future Improvements**

Building on the foundation of this research, several directions exist to enhance the performance, adaptability, and applicability of the system in broader communication contexts.

*7.4.1 Use of Real-Time SNR Estimation*

The reliance of the current implementation on indirect metrics such as ACK success rate and retry count as features to the neural network is what forces this reader to seek a better approach. Nevertheless, this decision quality would likely be improved by integrating a real time SNR estimation module on the hardware.

We can implement a Kalman filter or exponential moving average (EMA) estimator to compute these smoothed values of SNR from some computation of RSSI and Signal-to-Noise (SN) from the LoRa transceiver. These were more reliable values to be used as inputs to the FNN.

By doing so, the system to distinguish between the temporary packet drops (due to interference) and permanent deterioration of channels (caused by the distance or fading) could be improved in real time, and made decision of decoder selection with higher accuracy.

*7.4.2 Online Learning or Reinforcement Learning*

The primary future objective seeks to develop online learning systems from current offline-trained models. The approach consists of two alternatives to solve the objective.

During its field operations the device provides the FNN with capability to train itself by processing new packet statistics.

* The implementation of a lightweight Reinforcement Learning (RL) agent with Q-Learning or Deep Q-Network protocols would use transmission success and saved energy as reward factors.

The automated system develops stronger self-adaptive performance through adaptation to environmental changes while optimizing behavior across diverse operational conditions such as urban or rural deployments together with night or day interference.

The development of efficient learning algorithms faces two main challenges because the available memory and power resources are limited and may need experience replay compression or federated learning techniques to function effectively.

*7.4.3 Extension to LDPC and Turbo Codes*

Polar Codes are excellent for short-block 5G control channels, but longer payloads and certain IoT standards rely on other codes:

* **Low-Density Parity-Check (LDPC)**: Used in WiFi and 5G data channels, LDPC offers better BER performance but requires iterative decoding.
* **Turbo Codes**: Common in legacy LTE and some satellite systems.

Extending the decoder selection framework to these codes would require:

* Developing hardware-efficient implementations of their decoders.
* Extracting new feature sets (e.g., LLR distributions, convergence rate).
* Training larger FNNs or modular selectors for each code family.

This would make the system applicable to a wider variety of embedded devices and communication stacks, broadening its impact.

*7.4.4 Hardware Optimization for Ultra-Low Power*

To enable deployment on **coin-cell or energy-harvested devices**, further reduction in power consumption is necessary. Potential strategies include:

* **Custom Hardware Acceleration**: Implementing decoder logic and FNN inference using custom RTL in Verilog and porting it to FPGAs or ASICs.
* **Sleep Mode Optimization**: Dynamically switching the ESP32 between ultra-low-power and active modes with precise timers to minimize idle leakage.
* **TinyML Frameworks**: Migrating from TensorFlow Lite to smaller libraries like uTensor or CMSIS-NN that generate more efficient binaries.
* **Event-Driven Wakeups**: Use hardware interrupts from LoRa (e.g., DIO0) to trigger decoding only when data is available, avoiding polling and idle loops.

With these improvements, the system can be made viable for applications such as smart agriculture, wildlife monitoring, and disaster-resilient sensor networks where replacing batteries is infeasible

**Conclusion**

This research proved the approach to merge Polar Code decoding with lightweight neural networks for constructing smart energy-efficient communication components suitable for embedded platforms. Our system proved how automatic decoder selection methods could save considerable power while maintaining error levels unchanged so it became a crucial tool for present wireless systems.

Our work was aimed at designing and deploying a full stack implementation from simulation to neural network training to firmware to real hardware evaluation in order to play the basis for cognitive communication systems that can adapt in real time to their environment. The path to progress demands devices that possess learning capabilities together with decision-making power while also having autonomous energy-saving capabilities.

Among the many tradespace features of the new generation of energy efficient, intelligent wireless technologies, which appear promising from the vantage point of machine learning and embedded systems, this thesis explores wireless sensor networks (WSNs).

**REFERENCES**

* X. Liu and C. Huang, "Machine Learning for Polar Codes in Small IoT Devices," in Proceedings of the 2024 IEEE 7th International Conference on Big Data and Artificial Intelligence (BDAI), 2024, DOI: 10.1109/BDAI62182.2024.10692447.
* L. Huang, H. Zhang, R. Li, Y. Ge, and J. Wang, "AI Coding: Learning to Construct Error Correction Codes," IEEE Transactions on Communications, vol. 68, no. 1, pp. 1–12, Jan. 2020, DOI: 10.1109/TCOMM.2019.2955119.
* K. Almazrouei and K. A. Alnajjar, "Error-correcting Codes in Communication Systems," in Proceedings of the 20th International Wireless Communications and Mobile Computing Conference (IWCMC), 2024, pp. 1–10, DOI: 10.1109/IWCMC61514.2024.10592361.
* M. Ebada, S. Cammerer, A. Elkelesh, and S. ten Brink, "Deep Learning-based Polar Code Design," Institute of Telecommunications, University of Stuttgart, Stuttgart, Germany, 2024.
* E. Arıkan, "Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels," IEEE Transactions on Information Theory, vol. 55, no. 7, pp. 3051–3073, Jul. 2009, DOI: 10.1109/TIT.2009.2021379.
* S. Cammerer, J. Hoydis, and S. ten Brink, "Machine Learning for Communications: Current State and Future Directions," IEEE Communications Magazine, vol. 58, no. 6, pp. 14–19, Jun. 2020, DOI: 10.1109/MCOM.001.1900650.
  + Elkelesh, M. Ebada, S. Cammerer, and S. ten Brink, "Decoder-tailored Polar Code Design using the Genetic Algorithm," in Proc. of IEEE Vehicular Technology Conference (VTC-Fall), 2018, pp. 1–5, DOI: 10.1109/VTCFall.2018.8690832.
* S. Dörner, S. Cammerer, J. Hoydis, and S. ten Brink, "Deep Learning Based Communication Over the Air," IEEE Journal of Selected Topics in Signal Processing, vol. 12, no. 1, pp. 132–143, Feb. 2018, DOI: 10.1109/JSTSP.2017.2788002.
* R. T. Derryberry et al., "Advanced Receiver Techniques for 3G WCDMA," IEEE Communications Magazine, vol. 38, no. 7, pp. 134–143, Jul. 2000, DOI: 10.1109/35.852023.
  + - Goldsmith, Wireless Communications, 1st ed., Cambridge, U.K.: Cambridge University Press, 2005.
* S. Haykin, Neural Networks and Learning Machines, 3rd ed., Pearson Education, 2009.
* G. Fettweis and E. Zimmermann, "ICT Energy Consumption — Trends and Challenges," in Proceedings of the 11th International Symposium on Wireless Personal Multimedia Communications (WPMC), 2008.
* TensorFlow Lite for Microcontrollers Documentation. [Online]. Available: https://www.tensorflow.org/lite/microcontrollers
* Arduino IDE, Arduino.cc. [Online]. Available: https://www.arduino.cc/en/software
* Espressif Systems, “ESP32-WROOM-32 Datasheet,” [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32\_datasheet\_en.pdf
* Semtech Corporation, “SX1278/76/77/79 Datasheet – Long Range Transceiver,” [Online]. Available: https://www.semtech.com/uploads/documents/sx1276.pdf
* Numpy Developers, “NumPy: The fundamental package for scientific computing with Python,” [Online]. Available: <https://numpy.org>
* M. Abadi et al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015. [Software]. Available: <https://www.tensorflow.org/>
* Scikit-learn Developers, "Machine Learning in Python," [Online]. Available: <https://scikit-learn.org/>

# **Appendices**

## ****Appendix A: FNN Model Architecture****

The Feedforward Neural Network (FNN) used in this project was designed to perform multiclass classification to select the optimal Polar decoder (SC, SCL, or LVA) based on input communication features.

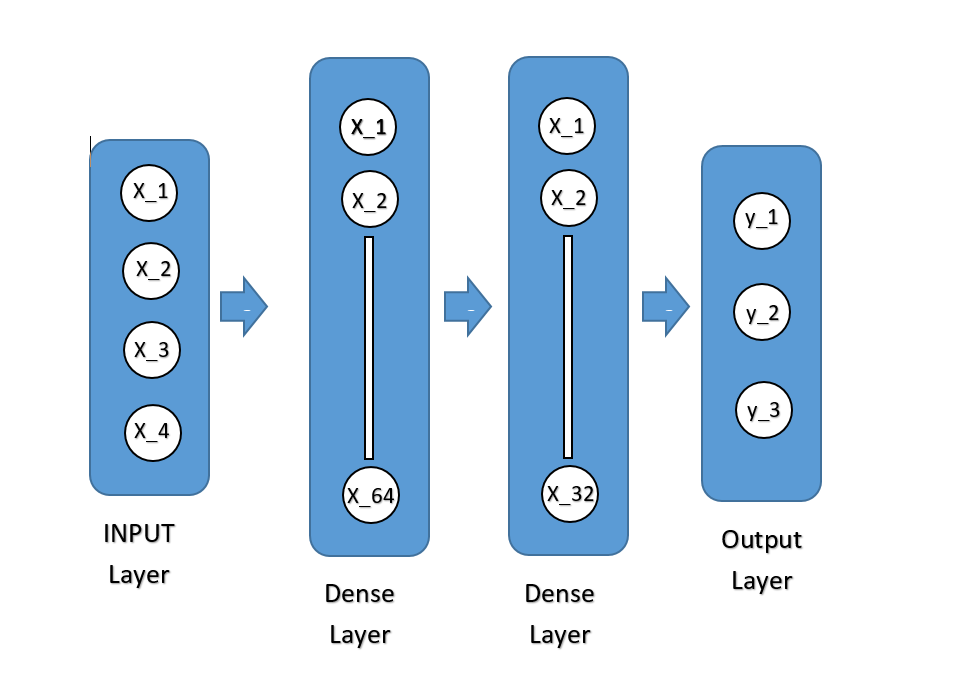
**Model Structure:**

* **Input Layer:** 4 neurons
  + Features:
    1. Signal-to-Noise Ratio (SNR)
    2. Packet Retry Count
    3. ACK Success Rate
    4. CRC Error Count
* **Hidden Layers:**
  + Layer 1: 64 neurons, ReLU activation
  + Layer 2: 32 neurons, ReLU activation
* **Output Layer:** 3 neurons, Softmax activation
  + Classes:
    1. Class 0: SC Decoder
    2. Class 1: SCL Decoder
    3. Class 2: LVA Decoder

**Training Details:**

* Loss Function: Categorical Cross-Entropy
* Optimizer: Adam (learning rate = 0.001)
* Epochs: 50
* Batch Size: 64
* Accuracy Achieved: ~92% on validation set

**Inference Time on ESP32:** ~5.8 ms using quantized TensorFlow Lite model



**Appendix B: Sample Code Snippets:**

1. **SC decoding:**

|  |
| --- |
| import numpy as np  from numpy.random import randn, randint  import matplotlib.pyplot as plt  from scipy.special import erfc  from tqdm import tqdm  # For the loading bar  # ----- Parameters ----- #  N = 1024               # Length of the polar codeword  K = 512                # Number of information bits  num\_bits = int(1e6)    # Total number of bits to simulate  design\_snr\_dB = 5      # Design SNR in dB for calculating frozen bits  snr\_db\_range = np.arange(0, 31, 1)  # SNR range from 0 to 30 dB  num\_trials = 100        # Number of trials per SNR value  ber\_threshold = 1e-2   # BER threshold for outage probability  # ----- Polar Code Construction ----- #  def generate\_kernel():      return np.array([[1, 0], [1, 1]])  def generate\_polar\_transform(n):      G = generate\_kernel()      for \_ in range(n - 1):          G = np.kron(G, generate\_kernel())      return G  def calculate\_bhattacharyya(N, design\_snr\_dB):      snr = 10\*\*(design\_snr\_dB / 10)      z = np.zeros(N)      z[0] = np.exp(-snr)      for lev in range(int(np.log2(N))):          B = 2\*\*lev          for i in range(B):              T = z[i]              z[2 \* i] = 2 \* T - T\*\*2  # Upper channel              z[2 \* i + 1] = T\*\*2      # Lower channel      return z  def get\_frozen\_bits(N, K, design\_snr\_dB):      z = calculate\_bhattacharyya(N, design\_snr\_dB)      indices = np.argsort(z)      info\_bits = np.sort(indices[:K])      frozen\_bits = np.ones(N, dtype=bool)      frozen\_bits[info\_bits] = False      return frozen\_bits  # ----- Polar Encoding ----- #  def polar\_encode(u, G\_N):      return np.mod(np.dot(u, G\_N), 2)  # ----- Recursive SC Decoder ----- #  def sc\_decode\_recursive(llr, frozen\_bits, u\_hat, depth=0):      N = len(llr)      if N == 1:          return np.array([0 if frozen\_bits[0] or llr[0] >= 0 else 1])        # LLR for left side      llr\_left = np.sign(llr[:N//2]) \* np.sign(llr[N//2:]) \* np.minimum(np.abs(llr[:N//2]), np.abs(llr[N//2:]))      u\_left = sc\_decode\_recursive(llr\_left, frozen\_bits[:N//2], u\_hat, depth + 1)        # LLR for right side      llr\_right = ((1 - 2 \* u\_left) \* llr[:N//2]) + llr[N//2:]      u\_right = sc\_decode\_recursive(llr\_right, frozen\_bits[N//2:], u\_hat, depth + 1)      return np.concatenate([u\_left ^ u\_right, u\_right])  def sc\_decode(llr, frozen\_bits):      u\_hat = np.zeros\_like(llr)      return sc\_decode\_recursive(llr, frozen\_bits, u\_hat)  # ----- Channel Simulations ----- #  def simulate\_rayleigh\_awgn(codeword, snr\_dB, frozen\_bits, G\_N):      snr = 10\*\*(snr\_dB / 10)      sigma = np.sqrt(1 / (2 \* snr))        # Rayleigh fading      h = np.random.rayleigh(scale=1.0, size=len(codeword))      bpsk = (1 - 2 \* codeword) \* h        # AWGN noise      noise = sigma \* randn(len(codeword))        # Received signal y = hX + noise      r = bpsk + noise        # Compute LLR and decode using SC decoder      llr = 2 \* r \* h / (sigma\*\*2)      u\_hat = sc\_decode(llr, frozen\_bits)      x\_hat = polar\_encode(u\_hat, G\_N)        return x\_hat  # ----- Main Simulation ----- #  def main():      G\_N = generate\_polar\_transform(int(np.log2(N)))      frozen\_bits = get\_frozen\_bits(N, K, design\_snr\_dB)        ber\_rayleigh\_awgn = []      outage\_probability = []      theoretical\_bpsk\_rayleigh\_awgn = []      print("Simulating... (can take a few minutes)")      for snr\_db in tqdm(snr\_db\_range, desc="SNR values", unit="dB"):          errors\_rayleigh\_awgn = 0          total = 0          outage\_count = 0            # 50 trials per SNR value          for trial in range(num\_trials):              # Random input bits              u = np.zeros(N, dtype=int)              info = randint(0, 2, K)              u[~frozen\_bits] = info              x = polar\_encode(u, G\_N)                # Simulate through Rayleigh fading and AWGN channel              x\_hat\_rayleigh\_awgn = simulate\_rayleigh\_awgn(x, snr\_db, frozen\_bits, G\_N)              total += K              errors = np.sum(x\_hat\_rayleigh\_awgn[~frozen\_bits] != info)              errors\_rayleigh\_awgn += errors                # Check if BER exceeds threshold for outage probability              if errors / K > ber\_threshold:                  outage\_count += 1          # Compute BER for Rayleigh + AWGN channel          ber\_rayleigh\_awgn.append(errors\_rayleigh\_awgn / total)            # Outage Probability          outage\_probability.append(outage\_count / num\_trials)            # Theoretical BPSK BER for Rayleigh + AWGN          snr\_linear = 10\*\*(snr\_db / 10)          theoretical\_bpsk\_rayleigh\_awgn.append(0.5 \* (1 - np.sqrt(snr\_linear / (snr\_linear + 1))))          print(f"SNR={snr\_db} dB | BER\_Rayleigh\_AWGN={ber\_rayleigh\_awgn[-1]:.4e} | Outage Probability={outage\_probability[-1]:.4f}")          ber\_rayleigh\_awgn = [max(ber, 1e-10) for ber in ber\_rayleigh\_awgn]      # Plotting BER vs. SNR      plt.figure(figsize=(10, 6))      plt.semilogy(snr\_db\_range, ber\_rayleigh\_awgn, 'bo-', label='Polar SC (Rayleigh + AWGN)')      plt.semilogy(snr\_db\_range, theoretical\_bpsk\_rayleigh\_awgn, 'g--', label='BPSK Rayleigh + AWGN (Theoretical)')      plt.xlabel("SNR (dB)")      plt.ylabel("Bit Error Rate (BER)")      plt.title("BER Performance of Polar Code (SC Decoder) with Rayleigh Fading + AWGN")      plt.grid(True, which='both')      plt.legend()      plt.tight\_layout()      plt.show()      # Plotting Outage Probability vs. SNR      plt.figure(figsize=(10, 6))      plt.plot(snr\_db\_range, outage\_probability, 'ro-', label='Outage Probability')      plt.xlabel("SNR (dB)")      plt.ylabel("Outage Probability")      plt.title("Outage Probability vs. SNR for Polar Code (SC Decoder) with Rayleigh Fading + AWGN")      plt.grid(True)      plt.legend()      plt.tight\_layout()      plt.show()  if \_\_name\_\_ == "\_\_main\_\_":      main() |

1. **SCL decoding:**

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  from tqdm import tqdm  # Parameters  N = 2048                     # Length of the codeword  K = 1024                     # Number of information bits  L = 16                        # SCL list size  snr\_db\_range = np.arange(0, 31, 1)  # SNR range in dB  design\_snr\_dB = 5            # Design SNR  total\_bits\_per\_snr = int(1e6) # Number of bits to simulate per SNR value  ber\_threshold = 1e-3         # BER threshold for outage  num\_trials = 150             # Number of trials per SNR  # Helper Functions  def log1pexp(x):      return np.where(x > 20, x, np.log1p(np.exp(x)))  def generate\_kernel():      return np.array([[1, 0], [1, 1]])  def generate\_polar\_transform(n):      G = generate\_kernel()      for \_ in range(n - 1):          G = np.kron(G, generate\_kernel())      return G  def calculate\_bhattacharyya(N, design\_snr\_dB):      snr = 10 \*\* (design\_snr\_dB / 10)      z = np.zeros(N)      z[0] = np.exp(-snr)      for lev in range(int(np.log2(N))):          B = 2\*\*lev          for i in range(B):              T = z[i]              z[2\*i] = 2\*T - T\*\*2              z[2\*i+1] = T\*\*2      return z  def get\_frozen\_bits(N, K, design\_snr\_dB):      z = calculate\_bhattacharyya(N, design\_snr\_dB)      indices = np.argsort(z)      info\_bits = np.sort(indices[:K])      frozen\_bits = np.ones(N, dtype=bool)      frozen\_bits[info\_bits] = False      return frozen\_bits  def polar\_encode(u, G\_N):      return np.mod(np.dot(u, G\_N), 2)  def polar\_scl\_decode(llr, frozen\_bits, L=4):      N = len(llr)      paths = [{'u': np.zeros(N, dtype=int), 'pm': 0.0}]      for i in range(N):          new\_paths = []          for path in paths:              u, pm = path['u'], path['pm']              llr\_val = llr[i]              if frozen\_bits[i]:                  bit = 0                  pm\_new = pm + log1pexp(np.abs(llr\_val))                  u\_new = np.copy(u)                  u\_new[i] = bit                  new\_paths.append({'u': u\_new, 'pm': pm\_new})              else:                  for bit in [0, 1]:                      if bit == 0:                          pm\_new = pm + log1pexp(llr\_val)                      else:                          pm\_new = pm + llr\_val + log1pexp(llr\_val)                      u\_new = np.copy(u)                      u\_new[i] = bit                      new\_paths.append({'u': u\_new, 'pm': pm\_new})          new\_paths.sort(key=lambda p: p['pm'])          paths = new\_paths[:L]      best\_path = min(paths, key=lambda p: p['pm'])      return best\_path['u']  def simulate(codeword, snr\_dB, frozen\_bits, G\_N, L):      snr = 10 \*\* (snr\_dB / 10)      sigma = np.sqrt(1 / (2 \* snr))      h = np.random.rayleigh(scale=1.0, size=len(codeword))      bpsk = (1 - 2 \* codeword) \* h      noise = sigma \* np.random.randn(len(codeword))      r = bpsk + noise      llr = 2 \* r \* h / (sigma\*\*2)      u\_hat = polar\_scl\_decode(llr, frozen\_bits, L)      x\_hat = polar\_encode(u\_hat, G\_N)      return x\_hat  def theoretical\_rayleigh\_ber(snr\_db):      snr\_linear = 10 \*\* (snr\_db / 10)      return 0.5 \* (1 - np.sqrt(snr\_linear / (1 + snr\_linear)))  # Main Simulation  def main():      G\_N = generate\_polar\_transform(int(np.log2(N)))      G\_N = np.mod(G\_N, 2).astype(int)      frozen\_bits = get\_frozen\_bits(N, K, design\_snr\_dB)      ber\_scl = []      outage\_probability = []      ber\_theory = theoretical\_rayleigh\_ber(snr\_db\_range)      print("Simulating Polar SCL over Rayleigh+AWGN")        for snr\_db in tqdm(snr\_db\_range, desc="SNR"):          total\_errors = 0          total\_blocks = 0          outage\_count = 0          for \_ in range(num\_trials):              u = np.zeros(N, dtype=int)              info\_bits = np.random.randint(0, 2, K)              u[~frozen\_bits] = info\_bits              x = polar\_encode(u, G\_N)              x\_hat = simulate(x, snr\_db, frozen\_bits, G\_N, L)              decoded\_info = x\_hat[~frozen\_bits]              errors = np.sum(decoded\_info != info\_bits)              total\_errors += errors              total\_blocks += 1              if errors / K > ber\_threshold:                  outage\_count += 1          ber = total\_errors / (total\_blocks \* K)          outage = outage\_count / num\_trials          ber\_scl.append(max(ber, 1e-10))          outage\_probability.append(outage)          print(f"SNR={snr\_db} dB | BER={ber:.4e} | Outage={outage:.4f}")      # Plotting BER vs SNR with Theoretical Comparison      plt.figure(figsize=(10, 6))      plt.semilogy(snr\_db\_range, ber\_scl, 'bo-', label='Simulated Polar SCL')      plt.semilogy(snr\_db\_range, ber\_theory, 'g--', label='Theoretical BPSK (Rayleigh)')      plt.xlabel("SNR (dB)")      plt.ylabel("Bit Error Rate (BER)")      plt.title("BER vs SNR: Polar SCL vs Theoretical Rayleigh BPSK")      plt.grid(True, which='both')      plt.legend()      plt.tight\_layout()      plt.show()      # Plotting Outage Probability vs SNR      plt.figure(figsize=(10, 6))      plt.plot(snr\_db\_range, outage\_probability, 'ro-', label='Outage Probability')      plt.xlabel("SNR (dB)")      plt.ylabel("Outage Probability")      plt.title("Outage Probability vs SNR")      plt.grid(True)      plt.legend()      plt.tight\_layout()      plt.show()  if \_\_name\_\_ == "\_\_main\_\_":      main() |

1. **LV decoding:**

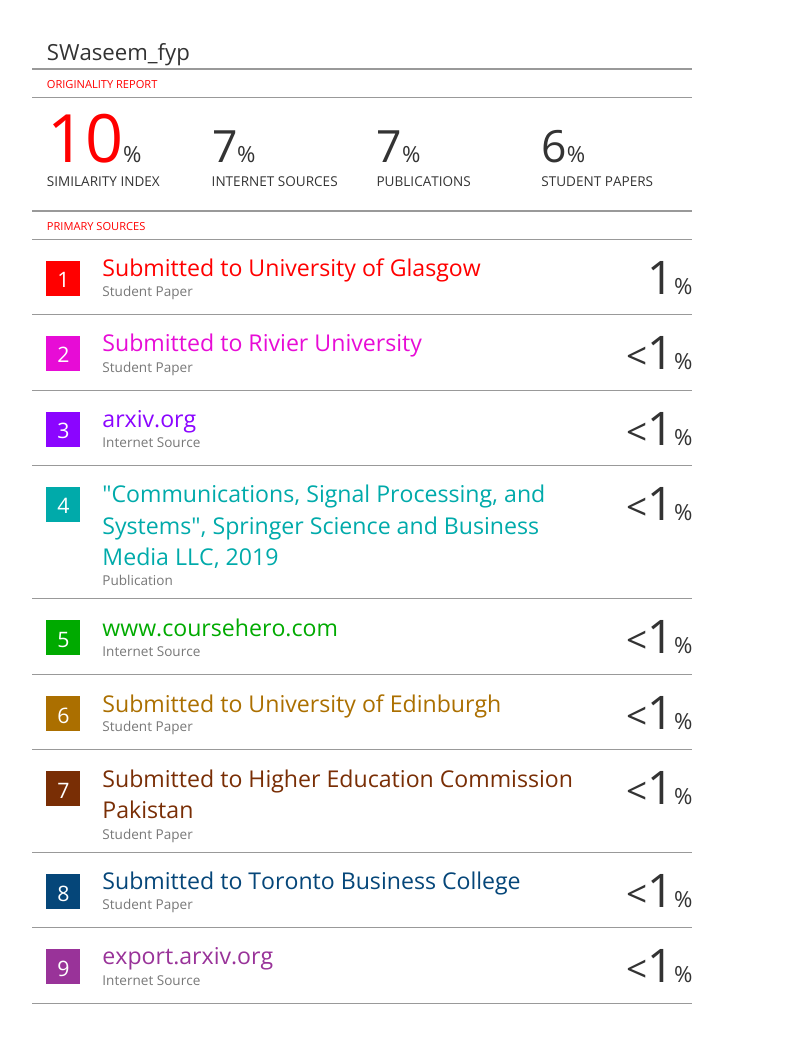
|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  from tqdm import tqdm  # Parameters  N = 2048                      # Length of the codeword  K = 1024                      # Number of information bits  L = 8                        # List size for LVA  snr\_db\_range = np.arange(0, 31, 1)  # SNR range in dB  design\_snr\_dB = 5             # Design SNR  total\_bits\_per\_snr = int(1e6) # Number of bits to simulate per SNR value  ber\_threshold = 1e-3          # BER threshold for outage  num\_trials = 100              # Number of trials per SNR (reduced for faster simulation)  # Helper Functions  def generate\_kernel():      return np.array([[1, 0], [1, 1]])  def generate\_polar\_transform(n):      G = generate\_kernel()      for \_ in range(n - 1):          G = np.kron(G, generate\_kernel())      return G  def calculate\_bhattacharyya(N, design\_snr\_dB):      snr = 10 \*\* (design\_snr\_dB / 10)      z = np.zeros(N)      z[0] = np.exp(-snr)      for lev in range(int(np.log2(N))):          B = 2\*\*lev          for i in range(B):              T = z[i]              z[2\*i] = 2\*T - T\*\*2              z[2\*i+1] = T\*\*2      return z  def get\_frozen\_bits(N, K, design\_snr\_dB):      z = calculate\_bhattacharyya(N, design\_snr\_dB)      indices = np.argsort(z)      info\_bits = np.sort(indices[:K])      frozen\_bits = np.ones(N, dtype=bool)      frozen\_bits[info\_bits] = False      return frozen\_bits  def polar\_encode(u, G\_N):      return np.mod(np.dot(u, G\_N), 2)  # Improved List Viterbi Algorithm (LVA) for Polar Codes  def improved\_list\_viterbi\_decode(r, h, sigma, frozen\_bits, G\_N, L=16):      N = len(r)      n = int(np.log2(N))        # LLR calculation for soft-decision input      llr = 2 \* r \* h / (sigma\*\*2)        # Preprocessing: Identify all information bit positions      info\_bit\_positions = np.where(~frozen\_bits)[0]        # Initialize the list of candidate paths with a single all-zero path      paths = [{'u': np.zeros(N, dtype=int), 'metric': 0.0}]        # Process each bit position      for i in range(N):          new\_paths = []            for path in paths:              u\_current = path['u'].copy()              current\_metric = path['metric']                # If frozen bit, we only have one option (bit = 0)              if frozen\_bits[i]:                  u\_current[i] = 0                  # Calculate the expected encoded bit                  # We use the property that for polar codes, the i-th bit of encoded x depends on u[0:i+1]                  x\_bit = np.mod(np.sum(u\_current[:i+1] @ G\_N[0:i+1, i]), 2)                    # Calculate log-likelihood contribution                  bit\_llr = llr[i]                  if x\_bit == 1:                      bit\_llr = -bit\_llr  # Flip sign if expected bit is 1                    # Update metric (sum of log-likelihoods, more negative is worse)                  # For Viterbi, we use negative log-likelihood as the metric (minimize)                  new\_metric = current\_metric - bit\_llr                    new\_paths.append({'u': u\_current, 'metric': new\_metric})              else:                  # For information bits, try both 0 and 1                  for bit in [0, 1]:                      u\_new = u\_current.copy()                      u\_new[i] = bit                        # Calculate the expected encoded bit                      x\_bit = np.mod(np.sum(u\_new[:i+1] @ G\_N[0:i+1, i]), 2)                        # Calculate log-likelihood contribution                      bit\_llr = llr[i]                      if x\_bit == 1:                          bit\_llr = -bit\_llr  # Flip sign if expected bit is 1                        # Update metric                      new\_metric = current\_metric - bit\_llr                        new\_paths.append({'u': u\_new, 'metric': new\_metric})            # Sort paths by metric (lower metric is better in our implementation)          new\_paths.sort(key=lambda p: p['metric'])            # Keep only the L best paths          paths = new\_paths[:L]        # Select the path with the best metric      best\_path\_idx = np.argmin([p['metric'] for p in paths])      u\_hat = paths[best\_path\_idx]['u']        # Re-encode to get the estimated codeword      x\_hat = polar\_encode(u\_hat, G\_N)        return x\_hat  def simulate(codeword, snr\_dB, frozen\_bits, G\_N, L):      snr = 10 \*\* (snr\_dB / 10)      sigma = np.sqrt(1 / (2 \* snr))      h = np.random.rayleigh(scale=1.0, size=len(codeword))      bpsk = (1 - 2 \* codeword) \* h      noise = sigma \* np.random.randn(len(codeword))      r = bpsk + noise        # Use improved List Viterbi Algorithm      x\_hat = improved\_list\_viterbi\_decode(r, h, sigma, frozen\_bits, G\_N, L)        return x\_hat  def theoretical\_rayleigh\_ber(snr\_db):      snr\_linear = 10 \*\* (snr\_db / 10)      return 0.5 \* (1 - np.sqrt(snr\_linear / (1 + snr\_linear)))  # Main Simulation  def main():      print("Generating polar transform matrix...")      G\_N = generate\_polar\_transform(int(np.log2(N)))      G\_N = np.mod(G\_N, 2).astype(int)        print("Determining frozen bit positions...")      frozen\_bits = get\_frozen\_bits(N, K, design\_snr\_dB)      ber\_lva = []      outage\_probability = []      ber\_theory = theoretical\_rayleigh\_ber(snr\_db\_range)      print("Simulating Polar with improved List Viterbi over Rayleigh+AWGN")        for snr\_db in tqdm(snr\_db\_range, desc="SNR"):          total\_errors = 0          total\_bits = 0          outage\_count = 0          for trial in range(num\_trials):              # Generate random information bits              info\_bits = np.random.randint(0, 2, K)                # Place information bits in non-frozen positions              u = np.zeros(N, dtype=int)              u[~frozen\_bits] = info\_bits                # Encode              x = polar\_encode(u, G\_N)                # Transmit and decode              x\_hat = simulate(x, snr\_db, frozen\_bits, G\_N, L)                # Extract decoded information bits              decoded\_info = x\_hat[~frozen\_bits]                # Count errors              bit\_errors = np.sum(decoded\_info != info\_bits)              total\_errors += bit\_errors              total\_bits += K                # Check for outage              if bit\_errors / K > ber\_threshold:                  outage\_count += 1                # Print progress for long simulations              if (trial + 1) % 10 == 0:                  current\_ber = total\_errors / total\_bits                  current\_outage = outage\_count / (trial + 1)                  print(f"SNR={snr\_db} dB | Trial {trial+1}/{num\_trials} | Current BER={current\_ber:.6e} | Current Outage={current\_outage:.4f}")          # Calculate final BER and outage probability          ber = total\_errors / total\_bits          outage = outage\_count / num\_trials            # Store results          ber\_lva.append(max(ber, 1e-10))  # Avoid log of zero in plot          outage\_probability.append(outage)          print(f"SNR={snr\_db} dB | Final BER={ber:.6e} | Final Outage={outage:.4f}")      print("Plotting results...")        # Plotting BER vs SNR with Theoretical Comparison      plt.figure(figsize=(10, 6))      plt.semilogy(snr\_db\_range, ber\_lva, 'bo-', label='Polar with List Viterbi')      plt.semilogy(snr\_db\_range, ber\_theory, 'g--', label='Theoretical BPSK (Rayleigh)')      plt.xlabel("SNR (dB)")      plt.ylabel("Bit Error Rate (BER)")      plt.title("BER vs SNR: Polar with List Viterbi vs Theoretical Rayleigh BPSK")      plt.grid(True, which='both')      plt.legend()      plt.tight\_layout()      plt.savefig('polar\_lva\_ber.png')      plt.show()      # Plotting Outage Probability vs SNR      plt.figure(figsize=(10, 6))      plt.semilogy(snr\_db\_range, outage\_probability, 'ro-', label='Outage Probability')      plt.xlabel("SNR (dB)")      plt.ylabel("Outage Probability")      plt.title("Outage Probability vs SNR (Threshold BER = 10^-4)")      plt.grid(True, which='both')      plt.legend()      plt.tight\_layout()      plt.savefig('polar\_lva\_outage.png')      plt.show()  if \_\_name\_\_ == "\_\_main\_\_":      main() |

1. **Model TRAINING:**

|  |
| --- |
| import pandas as pd  import numpy as np  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Input, Dropout  from tensorflow.keras.utils import to\_categorical  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import LabelEncoder  from sklearn.preprocessing import StandardScaler  import matplotlib.pyplot as plt  from tensorflow.keras.callbacks import EarlyStopping  from tensorflow.keras.optimizers import Adam  import sys  import io  # Force stdout encoding to UTF-8  sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')  # Load the dataset  data = pd.read\_csv('nrf24l01\_shuffled\_data.csv', encoding='utf-8')  # Assuming the last column is the target (SC, SCL, LV)  X = data.iloc[:, :-1].values  # All rows, all columns except the last one as features  y = data.iloc[:, -1].values   # All rows, last column as target  # Encode categorical labels (SC, SCL, LV) to integers  label\_encoder = LabelEncoder()  y\_encoded = label\_encoder.fit\_transform(y)  # Convert labels to one-hot encoding (softmax output)  y\_one\_hot = to\_categorical(y\_encoded, num\_classes=3)  # Split the data into training and testing sets (80-20 split)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_one\_hot, test\_size=0.2, random\_state=42)  # Feature scaling (optional, but can help improve performance)  scaler = StandardScaler()  X\_train = scaler.fit\_transform(X\_train)  X\_test = scaler.transform(X\_test)  # Create the model  model = Sequential()  # Input layer and first hidden layer  model.add(Input(shape=(4,)))  model.add(Dense(64, activation='relu'))  model.add(Dropout(0.7))  # Adding Dropout layer for regularization  # Second hidden layer  model.add(Dense(32, activation='relu'))  model.add(Dropout(0.7))  # Adding Dropout layer for regularization  # Output layer with softmax activation for multi-class classification  model.add(Dense(3, activation='softmax'))  # Compile the model  model.compile(loss='categorical\_crossentropy',                optimizer=Adam(learning\_rate=0.0001),                metrics=['accuracy'])  # Early stopping to prevent overfitting  early\_stopping = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)  # Train the model for 25 epochs  history = model.fit(X\_train, y\_train, epochs=25, batch\_size=32, validation\_data=(X\_test, y\_test), callbacks=[early\_stopping])  # Evaluate the model on the test set  test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test)  print(f'Test accuracy: {test\_accuracy\*100:.2f}%')  # Save the trained model (optional)  model.save('nrf24l01\_fnn\_model.h5')  # Visualizing the training history  plt.figure(figsize=(12, 6))  # Plot training & validation loss  plt.subplot(1, 2, 1)  plt.plot(history.history['loss'], label='Training Loss')  plt.plot(history.history['val\_loss'], label='Validation Loss')  plt.xlabel('Epochs')  plt.ylabel('Loss')  plt.title('Training and Validation Loss')  plt.legend()  # Plot training & validation accuracy  plt.subplot(1, 2, 2)  plt.plot(history.history['accuracy'], label='Training Accuracy')  plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')  plt.xlabel('Epochs')  plt.ylabel('Accuracy')  plt.title('Training and Validation Accuracy')  plt.legend()  # Show the plots  plt.tight\_layout()  plt.show()  # Additional evaluation using classification report and confusion matrix  from sklearn.metrics import classification\_report, confusion\_matrix  # Make predictions  y\_pred = model.predict(X\_test)  # Convert predictions from one-hot encoding to labels  y\_pred\_labels = np.argmax(y\_pred, axis=1)  y\_test\_labels = np.argmax(y\_test, axis=1)  # Print classification report and confusion matrix  print(classification\_report(y\_test\_labels, y\_pred\_labels, target\_names=label\_encoder.classes\_))  print("Confusion Matrix:")  print(confusion\_matrix(y\_test\_labels, y\_pred\_labels)) |
|  |

1. **MODEL TESTING:**

|  |
| --- |
| import pandas as pd  import numpy as np  from tensorflow.keras.models import load\_model  from sklearn.preprocessing import StandardScaler  from tensorflow.keras.utils import to\_categorical  from sklearn.preprocessing import LabelEncoder  from sklearn.metrics import confusion\_matrix, classification\_report  import sys  import io  # Force stdout encoding to UTF-8  sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')  # Load the trained model  model = load\_model('nrf24l01\_fnn\_model.h5')  # Load the dataset  data = pd.read\_csv('nrf24l01\_shuffled\_data.csv', encoding='utf-8')  # Select the unseen samples (8001 to 10000)  X\_unseen = data.iloc[8000:10000, :-1].values  # Rows 8001 to 10000 for features  y\_unseen = data.iloc[8000:10000, -1].values   # Rows 8001 to 10000 for target  # Encode categorical labels (SC, SCL, LV) to integers  label\_encoder = LabelEncoder()  y\_encoded = label\_encoder.fit\_transform(y\_unseen)  # Convert labels to one-hot encoding (softmax output)  y\_one\_hot = to\_categorical(y\_encoded, num\_classes=3)  # Feature scaling (apply the same scaler used in training)  scaler = StandardScaler()  X\_unseen = scaler.fit\_transform(X\_unseen)  # Evaluate the model on the unseen data  test\_loss, test\_accuracy = model.evaluate(X\_unseen, y\_one\_hot)  # Print the result  print(f'Test accuracy on unseen samples (8001 to 10000) from model evaluation: {test\_accuracy\*100:.2f}%')  # If you want to get predictions for the unseen data, use model.predict()  predictions = model.predict(X\_unseen)  # Convert predictions from one-hot encoded format to label  predicted\_labels = label\_encoder.inverse\_transform(np.argmax(predictions, axis=1))  # Print the predictions for the unseen samples  print('Predicted labels for unseen samples (8001 to 10000):')  print(predicted\_labels)  # Manually calculate accuracy by comparing predicted labels with actual labels  correct\_predictions = np.sum(predicted\_labels == y\_unseen)  accuracy\_manual = correct\_predictions / len(y\_unseen) \* 100  # Print the manually calculated accuracy  print(f'Manually calculated accuracy on unseen samples (8001 to 10000): {accuracy\_manual:.2f}%')  # Calculate the confusion matrix  cm = confusion\_matrix(y\_unseen, predicted\_labels)  # Print the confusion matrix  print("Confusion Matrix:")  print(cm)  # Get the classification report (precision, recall, f1-score)  report = classification\_report(y\_unseen, predicted\_labels, target\_names=label\_encoder.classes\_)  print("Classification Report:")  print(report) |

**­**