Graph Neural Networks for Channel Decoding - Course Project Deliverable

Anonymous submission

Abstract

In communication systems, achieving reliable data transmission over long distances or interference-prone environments often relies on Low-Density Parity-Check (LDPC) codes for error correction. This project explores Graph Neural Networks (GNNs) and their applications in both graph learning and channel decoding. I start by establishing foundational concepts of GNNs with coding examples to illustrate graph representation, node features, and message-passing. Using the Cora dataset—a benchmark where nodes represent documents and edges represent citations—I construct a GNN model for node classification to ground these ideas. Building on this, I apply the message-passing framework of GNNs to design a simple decoder for LDPC codes in PyTorch, demonstrating GNNs' suitability for efficient LDPC decoding. To validate the approach, I reimplement findings from a recent study on GNN-based LDPC decoding, evaluating performance in accuracy and robustness. This project provides a practical framework for using GNNs in both graph learning and error correction, underscoring their adaptability for structured data in communication systems.The project code is available at https://anonymous.4open.science/r/57000_AI-2172/

.

Introduction

In modern communication systems, reliable data transmission is essential, especially with growing demand for high-speed, quality data. Digital systems—whether for satellite, mobile, or internet communications—face challenges from noise, interference, and signal degradation, which can lead to errors and data loss. Channel coding, or error control coding, addresses this by adding redundancy to transmitted data, allowing the receiver to detect and often correct errors without retransmission, thus preserving data integrity.

Among error correction codes, Forward Error Correction (FEC) codes like Low-Density Parity-Check (LDPC) and Bose–Chaudhuri–Hocquenghem (BCH) codes are especially effective. LDPC codes use a sparse parity-check matrix to provide robust error correction, enabling efficient decoding close to the Shannon limit. BCH codes, designed for high reliability, can detect and correct multiple random errors, making them ideal for critical applications. Both types of FEC codes enhance system resilience, allowing accurate data transmission even in challenging, noisy environments.

Despite the effectiveness of traditional decoding methods, they have limitations, especially for short to medium-length codewords. For example, Belief Propagation (BP) decoding works well for long codewords with sparse graph structures, but it becomes less efficient for shorter codewords with dense structures. Maximum Likelihood (ML) decoding, although optimal, is computationally impractical for long codes due to its exponential complexity.

Graph Neural Networks (GNNs) have emerged as a promising solution to these challenges in decoding. GNNs are specialized neural networks that process data represented as graphs. In the context of channel coding, the parity-check matrix of a code can be represented as a graph where nodes represent variables (bits) and constraints (parity-checks), and edges represent the relationships between them. This representation allows GNNs to perform message passing over the graph, similar to BP but enhanced by learning from data.

The core idea of using GNNs for channel decoding is to train a neural network to learn a generalized message passing algorithm. Instead of using fixed, hand-crafted functions to update messages at each node, the GNN-based decoder learns these functions from data. This learning-based approach enables the GNN to adapt to complex structures and potentially outperform traditional decoding methods, especially for codes like BCH where BP decoding is suboptimal.

The current project deliverable involves a structured approach to understanding and implementing GNNs for applications in graph learning and error-correcting code decoding. Here's a breakdown of the approach:

1. **Foundational Understanding of GNNs**: The project begins by introducing key GNN concepts, including graph representation and message passing.
2. **Cora Dataset Node Classification**: UUsing the Cora dataset, a node classification task illustrates how GNNs apply to structured, real-world data.
3. **LDPC Decoder with GNN**: A simple GNN model for LDPC decoding is developed using PyTorch, showcasing how GNNs facilitate message-passing-based tasks.
4. **Reimplementing Research Results**: Key findings from NVIDIA's paper (Cammerer et al. 2022) are reimplemented, comparing the GNN decoder's performance with traditional BP decoders and uncoded transmissions.

Related Work

The field of channel coding has deep roots in information theory, with significant foundational work that laid the groundwork for modern coding methods. LDPC codes, introduced by (Gallager 1962), were among the earliest codes that allowed efficient error correction by exploiting sparsely connected graphs, creating a robust structure for reliable data transmission over noisy channels advancements in the graphical representation of codes were provided by G. D. Forney, who proposed normal realizations on graphs, demonstrating that codes could be efficiently decoded by passing messages on factor graphs in (Forney 2001). It is the foundational paper that introduces the concept of representing error-correcting codes using graphs. It explains how graphs can be utilized to un- derstand coding structures, which is a core idea in the current project work. In (Richardson and Urbanke 2008) presented a comprehensive treatment of LDPC and other modern coding theories, formalizing approaches that brought LDPC codes close to Shannon's capacity limit . In particular, (Chung et al. 2001) illustrated the potential of LDPC codes designed within 0.0045 dB of the Shannon limit, solidifying their role in channel coding applications.

Another pivotal developer correction came with the introduction of polar codes by (Arikan 2009), which offered a theoretically sound method for achieving capacity on symmetric binary-input memoryless channels. Polar codes leverage the concept of channel polarization and have become integral in modern communication standards, underscoring the importance of theoretically grounded, graph-structured codes for communication.

More recent work by (Varadarajulu et al. 2024) investigated GNN for BCH codes in satellite communication, illustrating the potential of GNNs to adapt to complex, high-noise environments with non-standard channel coding schemes using pooling techniques within GNNs to enhance de- coding performance. (Honkala, Korpi, and Huttunen 2021) extended these principles with DeepRX, a fully convolutional learning receiver that applies neural networks for end-to-end decoding, offering insights into deep learning architectures for signal processing tasks in modern communication systems.

Several studies have focused on the specific challenges of graph-based channel. For example, (Clausius et al. 2024) demonstrated the advantages of joint equalization and decoding using GNNs in highly correlated wireless channels, highlighting GNNs' adaptability to complex signal environments. combines equalization and decoding into a single GNN framework. Equalization is the process of correcting signal distortions before decoding. By integrating these two processes, the proposed method can improve overall system performance and reduce complexity. Similarly, (Liao et al. 2023) applied GNNs to the construction of polar codes, leveraging the graph structure of polar codes for optimized decoding performance. Furthermore, recent works on GNNs, such as the one by (Wu et al. 2023), proposes a method that uses shared parameters in the GNN for decoding. This means that instead of each node and edge in the graph needing its own set of parameters, the same parameters can be used across different parts of the network. This approach reduces the number of parameters needed, which makes the model lighter and easier to handle while still maintaining good performance. The authors show that this method achieves similar decoding accuracy with fewer resources compared to the proposed paper's approach.

The present work builds on these advancements, integrating GNNs within LDPC decoding. This project combines graph-based approaches with cutting-edge GNN techniques to explore the potential of GNNs in LDPC decoding tasks. By evaluating re-implementations of state-of-the-art GNN decoders, this study contributes to a growing body of research on the efficacy of machine learning in enhancing traditional error correction methods.

Problem Definition

The proposed project based on (Cammerer et al. 2022) aims to enhance error-correcting code decoding in wireless systems. As discussed, in real-world networks, data transmission errors occur due to noise and interference, requiring decoding to correct these mistakes. Traditional methods like BP and ML, while effective, have limitations, particularly with small message sizes or complex code structures. Therefore, the authors propose a GNN-based approach to improve decoding accuracy and efficiency across various codes and message lengths, addressing these challenges and aiming to create a universal framework to integrate channel code graphs into neural network designs.

Key Concepts in Problem Definition

Parity-Check Matrix (H): It is a fundamental matrix defines the structure of LDPC and BCH codes, where each row represents a parity-check constraint and each column corresponds to a bit in the codeword. Mathematically, a binary vector $\mathbf{c}$ represents a valid codeword if it satisfies $\mathbf{H}\cdot\mathbf{c}^T = \mathbf{0}$, where $\mathbf{H}$ is the parity-check matrix, $\mathbf{c}$ is the codeword vector, and $\mathbf{0}$ is the zero vector. This equation ensures that the codeword satisfies all parity-check constraints, making it a valid codeword in a Galois Field (GF). A sample parity-check matrix H is given by:

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Each column in $\mathbf{H}$ corresponds to a variable node (VN) v_i, representing one bit of the codeword, and each row corresponds to a check node (CN), representing a parity-check condition. Each entry is 0 or 1; a 1 at position (i, j) indicates that check node i is connected to variable node j in the Tanner graph of $\mathbf{H}$. The parity-check conditions from $\mathbf{H}$ are:

- **First row**: $v_1 + v_2 + v_4 = 0$ (mod 2)
- **Second row**: $v_2 + v_3 + v_5 = 0$ (mod 2)
- **Third row**: $v_1 + v_3 + v_6 = 0$ (mod 2)
- **Fourth row**: $v_3 + v_5 + v_6 + v_7 = 0$ (mod 2)

To be valid, a codeword must satisfy these parity-check equations, ensuring each selected group of bits sums to zero in modulo 2.

Tanner Graph: This bipartite graph represents the code structure, with VNs corresponding to transmitted bits and CNs representing parity-check constraints. Edges in the graph indicate connections dictated by non-zero elements in the parity-check matrix. The Tanner graph is formally defined as:

$$G = (V, C, E)$$

where V is the set of variable nodes (bits of the codeword), C is the set of check nodes (parity-check constraints), and E is the set of edges connecting variable nodes to check nodes. Each edge corresponds to a non-zero entry in the parity-check matrix $\mathbf{H}$, which enables iterative message-passing decoding. The tanner graph of the sample parity check matrix is given in figure 1.

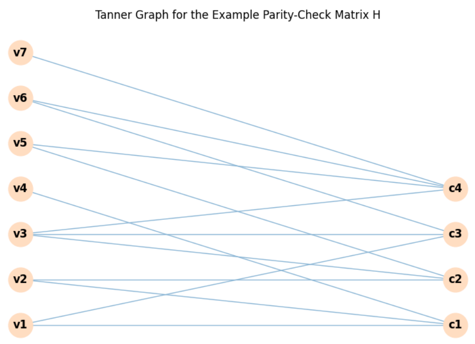

Figure 1: Tanner graph for the example parity-check matrix H. The code for generating this Tanner graph is available in the "Basic_LDPC_using_GNN.ipynb" notebook in the project link.

Log-Likelihood Ratio (LLR): A measure used to quantify the probability of each bit being a 0 or 1 based on received noisy data. The LLR is derived from the observed noisy signal y and is given by:

$$\text{LLR}(y) = \log \left(\frac{P(y|x = 0)}{P(y|x = 1)} \right)$$

where $P(y|x = 0)$ is the likelihood of receiving y given the transmitted bit $x = 0$, and $P(y|x = 1)$ is the likelihood of receiving y given the transmitted bit $x = 1$. The LLR is used to estimate the most likely transmitted bit, where a positive LLR suggests $x = 0$, and a negative LLR suggests $x = 1$.

Decoding Process Using Key Concepts

The GNN is trained on examples of transmitted and received data, allowing it to learn message-passing rules for error correction. Once trained, the GNN takes in noisy received data and applies its learned rules to estimate the most probable original message by iteratively updating estimates. The decoding process leverages the Parity-Check Matrix, Tanner Graph, and Log-Likelihood Ratio as follows:

1. Initialization with Log-Likelihood Ratio (LLR): For each received bit in $\mathbf{y}$, the initial probability of being 0 or 1 is calculated using the LLR. LLR values provide initial estimates for each VN in the Tanner graph, where a positive value suggests a higher likelihood of the bit being 0.

2. Message Passing in the Tanner Graph: The Tanner Graph, defined by the parity-check matrix $\mathbf{H}$, is a bipartite graph with VNs representing bits and CNs representing parity-check constraints. Connections in the graph specify which bits each parity-check involves. In message-passing: - **VN to CN**: Variable nodes send updated estimates to connected check nodes based on their LLR and incoming messages. - **CN to VN**: Check nodes send messages back to variable nodes, representing the probability that each bit satisfies the parity-check.

3. Parity-Check Constraints (Matrix H): Throughout iterations, variable nodes adjust estimates to satisfy the parity-check constraints. This ensures that all constraints are met for a valid codeword.

4. Final Decision: After iterations, a hard decision is made for each bit based on the final LLR values. Positive LLRs correspond to a bit value of 0, while negative LLRs correspond to 1, resulting in the decoded codeword that most likely represents the transmitted data.

Methodology

The main goal is to model the GNN and recreate the graphs discussed in the paper. The project was proposed as it will help me understand a new, unexplored neural network architecture and apply it to my main areas of study: wireless communications and signal processing. I can also as well explore the intersection of the fields with deep learning.

To approach the problem of decoding error-correcting codes with GNNs, it was essential first to understand the foundational principles of GNNs, given their relatively recent development and their unique ability to model and process graph-structured data. GNNs offer a promising alternative to traditional decoding techniques, as they can effectively learn and generalize from graph-like structures commonly found in parity-check matrices. Below, I present key concepts of GNNs that formed the foundation of my methodology.

Basic Concepts of GNNs

1. **Graph Representation and Nodes**: GNNs are neural networks designed for graph-structured data, with each graph G consisting of nodes V and edges E. Nodes in a GNN have feature vectors that store their information. In the decoding problem, each variable node in the Tanner graph could represent a bit, while each check node represents a parity-check constraint.
2. **Message Passing and Aggregation**: A core function of GNNs is message passing, where nodes communicate with neighbors to update their feature representations. In each iteration, a node gathers and aggregates information

from its neighbors, updating its own feature vector. This iterative process aligns with belief propagation in decoding, where information is exchanged along edges in the Tanner graph.

Message passing for a node v_i includes:

(a) **Message Computation**: Each node v_i computes messages for neighboring nodes using its own feature vector and those of its neighbors.

(b) **Aggregation**: A node aggregates messages from neighbors, typically using summing, averaging, or a similar function.

(c) **Update**: The node's feature vector is updated using the aggregated messages and its current state.
These steps for node v_i are summarized as:

$$h_i^{(t+1)} = \text{Update}\Big(h_i^{(t)}, \text{Aggregate}\Big(\big\{\text{Message}(h_i^{(t)}, h_j^{(t)}) \mid j \in \mathcal{N}(i)\big\}\Big)\Big) \quad (1)$$

where:

- $h_i^{(t)}$: the state (or feature) of node v_i at iteration t.
- $\mathcal{N}(i)$: the neighbors of v_i.
- Message, Aggregate, and Update are functions defined by the GNN architecture.

3. **Learning on Graphs**: The GNN is trained using supervised learning, where labeled data (e.g., noisy codewords and their true versions) guide it to learn effective message-passing. Through training, the GNN adjusts parameters to improve decoding accuracy.

Figure 2 illustrates message passing in a 5-node GNN. Each node exchanges information with neighbors over multiple iterations to update its features.

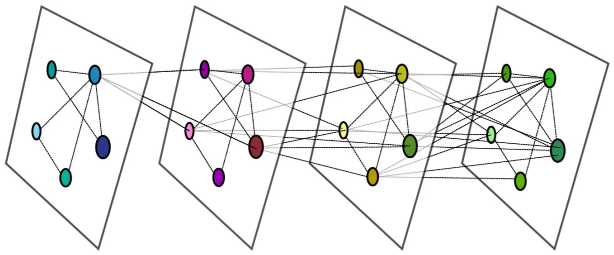

Figure 2: Message Passing Visualization in a 5-Node GNN. Each node aggregates information from its neighbors over multiple iterations.

By exploring foundational aspects of GNNs, I gained insights into designing, implementing, and training GNN models for decoding tasks. The file "*basic_GNN_concept.ipynb*" examines GNN operations using both PyTorch Geometric's built-in *MessagePassing* class and a manual PyTorch approach. A basic graph structure is defined, with each node initialized with features. The *SimpleGNNLayer* class leverages PyTorch Geometric's message-passing functions to aggregate and update node features, while the *ManualGNNLayer* implements these operations directly in PyTorch. This comparison clarifies GNN message-passing and feature transformation, laying a solid foundation for more complex Cora classification and LDPC decoding tasks.

The Cora dataset (McCallum et al. 2000), a widely used benchmark for graph learning, served to deepen my understanding of GNNs. In Cora, each node represents a scientific paper with features derived from word occurrences, and edges represent citation relationships. The goal is to classify nodes by category based on their features and neighboring nodes. To achieve this, I implemented a simple Graph Convolutional Network (GCN) model in "*GCN_classification_Cora.ipynb*". This GCN model includes two layers: the first layer applies ReLU activation after aggregating neighbor features, while the second layer uses log softmax for node classification. This exercise reinforced my understanding of GNN-based node classification.

Next, I explored applying GNNs to LDPC code decoding in "*basic_LDPC_using_GNN.ipynb*". Here, a Tanner graph was constructed from an LDPC parity-check matrix $\mathbf{H}$, forming a bipartite graph with variable nodes (representing codeword bits) and check nodes (parity-check constraints). The GNN iteratively refines predictions of codeword bits by message passing between connected nodes, simulating belief propagation. In this experiment, raw values of the received noisy bits served as input features, without using LLRs.

The experiment includes training, testing, and validation phases. During training, noise is added to codewords to simulate errors, and the GNN learns to correct them via message passing. Performance is evaluated through metrics like mean squared error (MSE) and bit mismatches. This setup illustrates the applicability of GNNs for decoding tasks, showing how message-passing techniques can aid in error correction in communication systems. This foundational work enabled us to replicate and extend key results from the authors' original paper, adapting significant portions of the code to a reduced training setup while maintaining effective model performance.

Key Steps in Paper Implementation

1. **Environment Setup and Package Imports**: Key packages like TensorFlow, PyTorch, and Sionna (for FEC utilities) were installed in Colab. Sionna's library from NVIDIA enables efficient channel simulation and LDPC decoding.

2. **End-to-End Model Definition**: An E2EModel class was created to simulate the communication pipeline. The model:

(a) Uses LDPC coding for error correction with parity-check matrices from Sionna.

(b) Integrates components like a Gaussian Prior Source and QAM Mapper/Demapper for symbol-based decoding.

(c) Tests different decoder setups, including uncoded, BP, and GNN-based LDPC decoding schemes.

3. **Baseline Decoding Methods**: Baseline decoding was implemented, starting with an uncoded scheme for initial BER/BLER reference, followed by BP-based LDPC decoding. Results provided a comparative baseline for GNN-based decoding performance.
4. **Weighted BP Model**: Based on (Nachmani, Be'ery, and Burshtein 2016), a Weighted BP model was developed to optimize BP decoding by introducing trainable weights. These weights are iteratively refined, and loss is averaged across decoding iterations for improved performance.
5. **GNN-Based Decoder Implementation**: The GNN-based BP decoder, the core of this project, leverages message-passing neural network layers for LDPC decoding. The model:
   (a) Facilitates message exchange between variable and check nodes in the Tanner graph.
   (b) Embeds nodes using multiple fully connected layers, enhancing representation power. Custom message-passing and update functions iteratively refine node embeddings, supporting effective belief propagation for decoding.
6. **Training and Model Checkpointing**: A custom training function was created to handle training and periodically save model weights. This process replicated the training structure from the original paper but scaled for reduced computational needs.

The outlined steps demonstrate a streamlined approach to implement and evaluate GNN-based LDPC decoding, adapting key methods from the authors' original framework.

Experimental Results and Result Analysis

For the Cora classification experiment, I trained the model for 200 epochs, tracking training, validation, and test accuracy to monitor its performance. The training accuracy rapidly achieved 100%, indicating that the model was able to fit the training data very well. The validation and test accuracy scores showed stable improvements during the training process, ultimately reaching a validation accuracy of around 78% and a test accuracy of 80.8%.

In the *"basic_LDPC_using_GNN.ipynb"* task the model was trained on a dataset of codewords with a focus on achieving zero bit mismatches for non tested codewords. The decoder includes three layers each for message and state update transformations. The message function is responsible for passing information from node to node, incorporating non-linear transformations with ReLU activations and a final sigmoid layer to handle belief propagation in a probabilistic manner. The update function, which aggregates and refines information, also applies multiple layers of transformations with ReLU activations. These layers enable the model to iteratively refine the node representations over a series of message-passing iterations (set by *num_iterations*), allowing for enhanced decoding accuracy across the graph structure. The training process was conducted over 20 epochs, where each epoch involved five iterations of message passing to enable information propagation through the graph structure. In each epoch, a batch of 10 noisy CWs was processed, allowing the model to learn the patterns and dependencies required for effective error correction.

The model's training performance, indicated by the average loss, showed variability across epochs, starting at a high average loss of 0.2017 in the first epoch, peaking at 0.4231 in the second epoch, and generally decreasing over time. By the final epoch (20), the model achieved an average loss of 0.1760, indicating improved decoding capability as training progressed. The iterative structure of the message-passing architecture, coupled with repeated exposure to the noisy codewords, allowed the model to progressively reduce errors and enhance accuracy, demonstrating the potential of GNN-based decoders for channel coding tasks. The testing output is shown in Table 1.

Table 1: Decoded Codewords with True and Predicted Values, Iterations, MSE, and Bit Mismatches

True Codeword	Predicted Codeword	Iteration	MSE	Bit Mismatches
[1 1 1 1 0 0 1 0 0 1 1 1 0 0 1 1]	[1 1 1 1 0 0 1 0 0 1 1 1 0 0 1 1]	1	0.0304	0
[0 1 0 1 0 1 0 0 1 0 1 1 0 0 1 1]	[0 1 0 1 0 1 0 0 1 0 1 1 1 0 1 1]	2	0.0634	1
[0 0 0 1 1 0 0 1 1 0 1 1 0 1 0 1]	[0 0 0 1 1 0 0 1 1 0 1 1 0 1 0 1]	1	0.0683	0
[0 0 0 1 1 0 1 0 0 0 1 0 1 1 1 1]	[0 0 0 0 1 0 1 0 0 0 1 0 1 1 1 1]	1	0.1259	1
[0 1 0 0 1 1 0 1 0 0 0 0 0 1 1 0]	[0 1 0 0 1 1 0 1 0 0 0 0 0 1 1 0]	1	0.0357	0
[0 0 1 0 1 1 0 0 0 0 0 0 1 1 1 0]	[0 0 1 0 1 1 0 0 0 0 0 0 1 1 1 0]	1	0.0377	0
[1 0 0 0 1 1 0 0 1 1 1 1 0 0 0 1]	[0 0 0 0 1 1 0 0 1 1 1 1 0 0 0 1]	1	0.0774	1
[0 1 0 1 0 0 1 1 0 0 1 0 1 0 0 0]	[0 0 0 1 0 0 1 1 0 0 1 0 1 0 0 0]	1	0.1493	1
[1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 1]	[1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0]	2	0.1109	1
[1 1 1 1 1 0 1 0 1 0 0 0 1 0 0 0]	[1 1 1 0 1 0 1 0 1 0 0 0 1 0 0 0]	1	0.1065	1

In the training setup for the paper reimplementation, the GNN-based LDPC decoder model is trained using a staged approach across three training phases, each with decreasing learning rates for gradual optimization. Each phase operates at different SNR levels, starting from 5.0 dB in the initial phase and increasing by 0.5 dB for each subsequent stage to improve the model's generalization. The training parameters are adjusted to 3,500 iterations for Phase 1, and 10,000 iterations each for Phases 2 and 3, resulting in a total of 23,500 iterations with progressively smaller learning rates of 1×10^{-3},1×10^{-4} and 1×10^{-5} respectively. This setup allows the model to gradually refine its decoding accuracy by balancing exploration and fine-tuning A final evaluation on BLER and BER is performed across different SNR values, measuring decoding performance and convergence as compared to the original implementation in the authors' baseline model. The observed GNN based decoding results is shown in figure 3 and a comparison detailing my implementation with the authors' original implementation is provided in Table 2.

In this reimplementation, the GNN-based decoder was trained with a reduced number of iterations compared to the extensive training in the authors' original implementation. Specifically, while the authors' model underwent 635,000 training iterations across multiple phases, the reduced setup employed only 20,000 iterations. Despite the reduced training, the reimplementation achieved a competitive error correction performance, demonstrating the resilience and efficiency of GNN-based decoders even with limited training. This reduction not only decreases computational requirements and training time but also makes the model more accessible for experimentation and adaptation in real-time or resource-constrained environments.

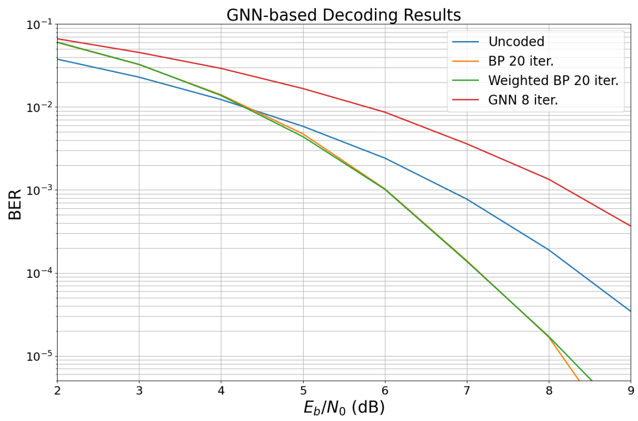

Figure 3: Decoding performance results using the reimplemented GNN-based model.

Conclusion and Future Directions

In conclusion, this project explores the application of Graph Neural Networks (GNNs) to the decoding of error-correcting codes, specifically focusing on the use of GNNs for decoding Low-Density Parity-Check (LDPC) and other structured codes. By implementing and testing a GNN-based decoder, we gained a deeper understanding of GNN message-passing mechanisms and their adaptability to channel coding challenges. The reimplementation demonstrated that even with reduced training iterations, the GNN-based decoder maintained competitive performance, indicating the potential for resource-efficient, high-performing decoders in practical scenarios. Key findings from this work reveal that GNNs can leverage the structural properties of LDPC codes, as seen in the Tanner graph representation, to enhance decoding accuracy and speed. This efficiency allows for notable reductions in training time and computational demand compared to more traditional decoding algorithms. Additionally, my results suggest that GNNs, when trained on limited iterations, can still generalize well for decoding, making them attractive for applications in real-time and edge-computing contexts.

For future work, several promising directions are identified. First, integrating additional GNN architectures, such as graph transformers or attention-based models, could enhance decoding accuracy by capturing complex dependencies more effectively. Second, extending the GNN framework to other structured codes, such as BCH and polar codes, may demonstrate broader applicability across different coding scenarios. Finally, investigating unsupervised or self-supervised training strategies could further reduce the need for extensive labeled data, making GNN-based decoders more versatile in various practical deployments. Through these future directions, GNNs have the potential to become a cornerstone of adaptive, scalable solutions in communication systems.

References

Arikan, E. 2009. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on information Theory*, 55(7): 3051–3073.

Cammerer, S.; Hoydis, J.; Aoudia, F. A.; and Keller, A. 2022. Graph neural networks for channel decoding. In *2022 IEEE Globecom Workshops (GC Wkshps)*, 486–491. IEEE.

Chung, S.-Y.; Forney, G. D.; Richardson, T. J.; and Urbanke, R. 2001. On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit. *IEEE Communications letters*, 5(2): 58–60.

Clausius, J.; Geiselhart, M.; Tandler, D.; and ten Brink, S. 2024. Graph Neural Network-Based Joint Equalization and Decoding. In *2024 IEEE International Symposium on Information Theory (ISIT)*, 1203–1208. IEEE.

Forney, G. D. 2001. Codes on graphs: Normal realizations. *IEEE Transactions on Information Theory*, 47(2): 520–548.

Gallager, R. 1962. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1): 21–28.

Honkala, M.; Korpi, D.; and Huttunen, J. M. 2021. DeepRx: Fully convolutional deep learning receiver. *IEEE Transactions on Wireless Communications*, 20(6): 3925–3940.

Liao, Y.; Hashemi, S. A.; Yang, H.; and Cioffi, J. M. 2023. Scalable polar code construction for successive cancellation list decoding: A graph neural network-based approach. *IEEE Transactions on Communications*.

McCallum, A. K.; Nigam, K.; Rennie, J.; and Seymore, K. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3: 127–163.

Nachmani, E.; Be'ery, Y.; and Burshtein, D. 2016. Learning to decode linear codes using deep learning. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 341–346. IEEE.

Richardson, T.; and Urbanke, R. 2008. *Modern coding theory*. Cambridge university press.

Varadarajulu, S.; Baeza, V. M.; Querol, J.; Mendonca, M. O.; and Chatzinotas, S. 2024. Graph Neural Network Pooling for BCH Channel Decoding in Software Defined Satellites. In *2024 IEEE International Conference on Communications Workshops (ICC Workshops)*, 1328–1333. IEEE.

Wu, Q.; Ng, B. K.; Lam, C.-T.; Cen, X.; Liang, Y.; and Ma, Y. 2023. Shared Graph Neural Network for Channel Decoding. *Applied Sciences*, 13(23): 12657.

Table 2: Comparison of BER and BLER between Reduced Training and Original Training Setups

EbNo [dB]	**BER (Mine)**	**BER (Authors)**	**Diff. BER**	**BLER (Mine)**	**BLER (Authors)**	**Diff. BLER**	**Comments**
2.0	6.66e-02	5.78e-02	+1.88e-02	9.89e-01	7.55e-01	+2.33e-01	Slightly higher BER/BLER in the reduced training setup, but results are still quite close to the authors' model. This indicates the GNN's ability to retain decoding effectiveness even with fewer training iterations at low SNRs.
3.0	4.55e-02	2.87e-02	+1.68e-02	9.45e-01	4.58e-01	+4.87e-01	A more noticeable increase in BER/BLER as SNR improves. The results suggest that while reduced training provides reasonable decoding, the model could benefit from additional training for improved accuracy at this SNR level.
4.0	2.92e-02	1.01e-02	+1.91e-02	8.47e-01	1.90e-01	+6.56e-01	At 4 dB, BER and BLER show increased differences, highlighting the impact of limited training. However, the model maintains robustness in decoding, indicating practical viability despite higher error rates in the mid-SNR range.
5.0	1.67e-02	2.01e-03	+1.47e-02	6.57e-01	4.52e-02	+6.12e-01	The reduced training model shows significantly higher BER/BLER compared to the authors' model. This suggests a trade-off, where reduced training saves resources but impacts performance. Still, it remains viable for moderate SNR applications.
6.0	8.67e-03	2.28e-04	+8.44e-03	4.21e-01	6.12e-03	+4.15e-01	As SNR increases, the gap between reduced and original training becomes more pronounced. Additional training iterations could potentially narrow this gap, especially for applications requiring high accuracy at elevated SNR levels.
7.0	3.60e-03	1.69e-05	+3.58e-03	2.03e-01	5.75e-04	+2.03e-01	Higher error rates with reduced training indicate potential room for improvement, especially in high-SNR ranges. Further fine-tuning or adaptive learning could enhance decoding reliability under these conditions.
8.0	1.34e-03	1.87e-06	+1.34e-03	8.11e-02	8.60e-05	+8.10e-02	At this SNR, the performance gap continues to be significant. The reduced training setup achieves moderate success, though further optimization is needed for high-precision applications in this range.
9.0	3.67e-04	2.38e-07	+3.67e-04	2.29e-02	1.40e-05	+2.28e-02	A large performance gap exists at 9 dB, suggesting that reduced training may not fully capture the required complexity at high SNR levels. Despite this, the model demonstrates proof-of-concept efficiency for simpler decoding tasks with lower computational demand.