

A Layer-Wise Syndrome-Based Framework for LDPC Decoding with Deep Learning

Ziwen Ren^{1,+(✉)} and Pengcheng Wang^{1,+}

¹ East China Normal University, Shanghai 200062, China

⁺ These authors contribute equally to this work
51265902060@stu.ecnu.edu.cn

Abstract. This paper proposes a layer-wise syndrome-based deep learning framework for decoding Low-Density Parity-Check (LDPC) codes. By integrating the Normalized Offset Min-Sum (NOMS) algorithm into a Forward-Feed-back Neural Network (NOMS-FF), the proposed model introduces a soft-syndrome loss function to enforce parity-check consistency and optimize decoding performance across varying noise levels. To further exploit the distinct error distributions across unrolled iterations, the framework incorporates a targeted post-training methodology utilizing layer-freezing and dynamic Signal-to-Noise Ratio (SNR) allocation. Extensive experiments on 6G-candidate QC-LDPC codes across multiple code lengths (from $N = 240$ to $N = 4032$) demonstrate that the proposed approach achieves substantial performance advantages over traditional decoders and state-of-the-art deep learning baselines. Furthermore, partial-layer decoding experiments reveal that deploying neural network updates exclusively in the final iterations, combined with global fine-tuning, not only reduces computational complexity but also completely suppresses the error floor, yielding unprecedented decoding accuracy at high SNRs.

Keywords: Deep Learning, Low-Density Parity-Check Codes, Belief Propagation, Normalized Offset Min-Sum, Soft-Syndrome Loss, Layer-Freezing, Dynamic SNR Allocation.

1 Introduction

In recent years, deep learning has been applied extensively to various research domains, including communication systems, drawing considerable research interest. Early foundational and highly influential studies, such as [5, 20], introduced the use of neural networks for error-correcting codes and convolutional decoding. These works directly used codewords as both input and output, leveraging neural networks for decoding. However, these early approaches encountered specific structural challenges in terms of performance and scalability, which limited their ability to consistently surpass traditional decoding algorithms.

With the continuous advancements in deep learning and communication coding research, advanced strategies have been explored to address these limitations. In the do-

main of short codes, deep learning-based decoding models have demonstrated substantial methodological advancements, as shown by works such as [11], [17], [21], [13], and [16], which explored various architectures including polar decoders, residual networks, and unfolded belief propagation to improve upon baseline traditional methods in decoding performance for short codes. However, these methods also mainly relied on directly using codewords as input and output. As noted in [11], when these techniques were extended to longer codes, traditional training methods often led to disproportionately large increases in model complexity, thereby inherently limiting their practical applicability.

Researchers proposed the idea of employing syndromes as a substitute input representation in place of codewords in order to overcome these difficulties. In [4], Bennatan et al. presented a deep learning framework based on syndromes and showed that using the structural information of syndromes improved model scalability and decoding performance, especially for long codes. This method established the foundation for combining conventional error correcting techniques with deep learning.

Building on earlier studies of deep learning-based decoding models that investigated several kinds of codewords, Low-Density Parity-Check (LDPC) codes have received special attention. As the telecommunications industry moves toward 6th Generation (6G) networks, LDPC codes are being used as a foundational benchmark for evaluating next-generation channel coding and AI-assisted decoding techniques [18, 19]. Belief propagation (BP) decoding is a well-known and efficient decoding technique for LDPC codes. Through check nodes (CNs) and variable nodes (VNs), which correspond to parity-check equations and codeword bits, respectively, BP decoding operates iteratively on a Tanner graph that depicts the structure of LDPC codes. Numerous optimization strategies have been put out to further improve BP decoding efficiency. For example, the min-sum (MS) decoding algorithm is produced by simplifying CN updates using the min-sum approximation [9]. Nevertheless, there is a trade-off with decoding performance as a result of this reduction. To address this issue, normalized min-sum (NMS) and offset min-sum (OMS) decoders [6] were created, which strike a balance between computational complexity and decoding accuracy.

Due to the structural similarity between Tanner graphs and neural networks, many researchers have designed neural network-based decoders inspired by Tanner graph structures. For instance, in [15], the authors proposed an unrolled BP network, treating BP decoding as a differentiable process, where decoding messages are passed iteratively in a feedforward manner.

Trainable weights were introduced on the edges of the factor graph and optimized via stochastic gradient descent (SGD) on unrolled iterative BP decoding.

Building on this foundation, in [16], the authors proposed neural network-based NMS and OMS decoders, which reduced the complexity of BP decoders and generalized the standard NMS and OMS methods [6]. Additionally, in [22], researchers introduced a neural network-aided normalized offset min-sum (NOMS) decoding method tailored for layered BP decoding, which has been applied to 5G LDPC codes.

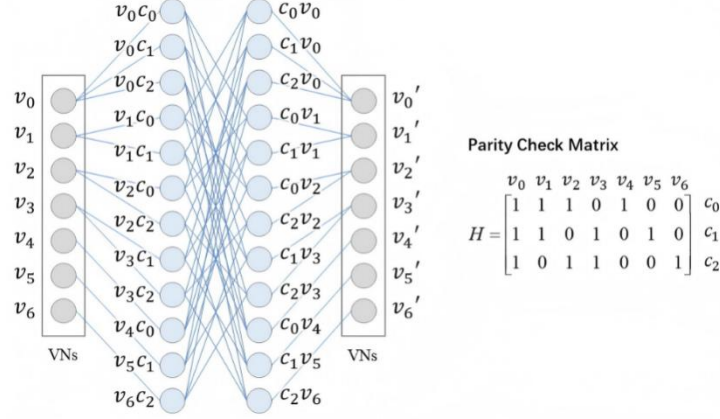


Fig. 1. An example of an unrolled BP decoder.

Fig. 1 illustrates the unrolled BP network proposed for a (7,3) LDPC code. The left side of the figure shows the unrolled BP network representing one iteration of the decoding process, while the right side depicts the corresponding parity-check matrix. In this network, the input and output layers each contain $N = 7$ neurons, while the hidden layer contains L neurons, where L is the total number of ones in the parity-check matrix, corresponding to the edges in the Tanner graph. This architecture forms a partially connected neural network.

We observed that another approach leveraging syndrome to optimize channel decoders has provided measurable improvements in the field of channel decoding. In [14], the authors proposed a loss function based on soft syndrome, which was used to train a neural network version of the BP decoder. Unlike [4], which employs hard syndrome as the decoder input, this approach introduced soft syndrome alongside the traditional cross-entropy loss function, enabling the model to capture the structural characteristics of codewords.

Building on recent advances in deep learning for decoding LDPC codes, this study proposes a technique to alleviate performance limits in traditional decoders by combining syndrome-based loss functions with a layer-wise training perspective. The primary contributions of our work are summarized as follows.

Inspired by [14], we found that syndrome-based loss functions can structurally guide the model to learn the complete structural features of codewords. By leveraging this capability, we conducted comprehensive research and corresponding experiments on 6G-candidate QC-LDPC codes, yielding performance that exceeds that of baseline traditional LDPC decoders.

To improve the decoding performance of LDPC codes, we used the NOMS network structure proposed in [22] to construct a NOMS-FF (Forward-Feedback Variant) decoder specifically for LDPC codes. We created a syndrome-based loss function during training by consulting the soft-syndrome techniques suggested in [1] and [14]. This design allowed us to address the conditions posed by varying noise levels during decoding. Specifically, we observed the iterative nature of BP decoders, where earlier

layers handle high-noise codewords close to the raw channel output, while later layers deal with reduced noise due to iterative decoding.

To push decoder performance further, we proposed a structured post-training framework from a layer-wise perspective. This framework leans on a selective layer-locking strategy—freezing certain layers while recalibrating the Signal-to-Noise Ratio (SNR) distribution of the training data for those remaining active. Such a design moves away from "one-size-fits-all" training, allowing for targeted, layer-specific optimization that aligns with the unique noise characteristics at different depths of the network.

Experimental data confirmed the utility of this approach. The post-training framework effectively irons out the convergence snags and performance caps that often plague the initial training phase. Moreover, the post-trained model can decode using only a subset of its layers, and under specific noise conditions, this partial-layer decoding strategy outperformed full-model decoding, achieving competitive performance within its corresponding noise range. These findings show the model's flexibility and effectiveness in managing diverse noise settings.

The paper is organized as follows. Section 2 introduces the structure of the model, focusing on how the soft-syndrome strategy is incorporated into the loss function to optimize decoding performance. It also describes the design and implementation of the layer-wise post-training framework. Section 3 presents and analyzes the experimental results, evaluating the performance of the NOMS-FF model and demonstrating the effectiveness of the proposed post-training framework and partial-layer decoding under different noise conditions.

2 Layer-Wise Syndrome-Based Decoder

2.1 System Model

In this work, we consider a linear block code C with code length n and information word length k , resulting in a code rate $R = k/n$. Let $G \in \mathbb{F}_2^{n \times k}$ and $H \in \mathbb{F}_2^{(n-k) \times n}$ represent the generator matrix and parity-check matrix of the code, respectively. An information word $\mathbf{u} \in \{0,1\}^k$ is encoded into a codeword $\mathbf{c} \in \{0,1\}^n$ using:

$$\mathbf{c} = G\mathbf{u}. \quad (1)$$

The encoded codeword $\mathbf{c}$ is modulated using binary phase-shift keying (BPSK), mapping each bit to +1 or -1, resulting in the modulated signal $\mathbf{x} \in \{-1, +1\}^n$:

$$x_i = \begin{cases} +1, & \text{if } c_i = 0, \\ -1, & \text{if } c_i = 1, \end{cases} \quad \forall i \in \{1, \dots, n\}. \quad (2)$$

The modulated signal $\mathbf{x}$ is transmitted over an Additive White Gaussian Noise (AWGN) channel, where the received signal $\mathbf{y} \in \mathbb{R}^n$ is given by:

$$\mathbf{y} = \mathbf{x} + \mathbf{z}, \quad (3)$$

with $\mathbf{z} \sim \mathcal{N}(0, \sigma^2 \mathbf{I}_n)$ being the noise vector. The parity-check constraints of the code are defined by the parity-check matrix H , and the syndrome $\mathbf{s} \in \mathbb{F}_2^{n-k}$ is calculated as:

$$\mathbf{s} = H\mathbf{c}^\top. \quad (4)$$

The decoding performance is evaluated in terms of the bit error rate (BER), which is defined as:

$$\text{BER} = \frac{1}{k} \sum_{i=1}^k \Pr(u_i \neq \hat{u}_i), \quad (5)$$

where $\hat{\mathbf{u}}$ is the estimated information word after decoding.

2.2 Syndrome-Based Approach

To support the performance of the decoder during the post-training phase, this paper introduces a modified loss function that incorporates layer-wise dynamically normalized regularization. By leveraging exponential moving averages, a standard regularization technique in deep learning [10], for inter-layer weight regularization, this approach ensures that the regularization constraints across layers remain balanced during training, while maintaining stability and generalization in the post-training phase.

During the initial training phase, the loss function only includes the CrossEntropyLoss and SyndromeLoss, and is defined as:

$$\mathcal{L}_{\text{initial}} = \lambda_1 \cdot \mathcal{L}_{\text{CE}} + \lambda_2 \cdot \mathcal{L}_{\text{syn}}, \quad (6)$$

where: $\mathcal{L}_{\text{CE}}$ minimizes the bit-level error between the predicted output and the ground truth; $\mathcal{L}_{\text{syn}}$ enforces parity-check consistency, defined as:

$$\mathcal{L}_{\text{syn}} = \frac{1}{n-k} \sum_{i=1}^{n-k} \|\tilde{s}_i\|^2, \quad (7)$$

where $\tilde{s}_i$ represents the soft-syndrome value for the i -th check node.

During the post-training phase, a layer-wise regularization term is introduced to further constrain the weights of unfrozen layers. The complete loss function is defined as:

$$\mathcal{L}_{\text{post}} = \lambda_1 \cdot \mathcal{L}_{\text{CE}} + \lambda_2 \cdot \mathcal{L}_{\text{syn}} + \lambda_3 \cdot \mathcal{R}_{\text{layer}}, \quad (8)$$

where: $\mathcal{R}_{\text{layer}}$ is the layer-wise normalized regularization term, defined as:

$$\mathcal{R}_{\text{layer}} = \frac{1}{|\mathcal{F}|} \sum_{l \in \mathcal{F}} \frac{\alpha_l \|\mathbf{W}_l\|^2}{\max(\bar{r}, \epsilon)}, \quad (9)$$

where $\mathcal{F}$ represents the set of unfrozen layers, and $\mathbf{W}_l$ is the weight matrix of the l -th layer; α_l is a layer-wise decreasing scaling factor, such as $\alpha_l = \frac{1}{l+1}$; $\bar{r}$ is the exponential moving average of the regularization value, initialized for the first layer and updated as:

$$\bar{r}_{l+1} = 0.9 \cdot \bar{r}_l + 0.1 \cdot \alpha_l \|\mathbf{W}_l\|^2. \quad (10)$$

This design ensures that the regularization constraints across layers are dynamically balanced. Specifically, shallow layers receive stronger regularization to stabilize the model's baseline feature extraction capabilities, while deeper layers receive weaker

constraints to allow them to adapt to complex noise distributions, supporting robustness under low SNR conditions.

To balance the optimization objectives between the initial training and post-training phases, the weights λ_1, λ_2 , and λ_3 in the loss function are dynamically adjusted as follows:

$$\lambda_1(t) = 1 - \frac{t}{T}, \lambda_2(t) = \frac{t}{T}, \lambda_3 = \text{constant}, \quad (11)$$

where t is the current training epoch, and T is the total number of epochs. This strategy allows the initial training phase to prioritize bit-level accuracy, while the post-training phase reinforces parity-check consistency and model stability.

The proposed loss function design has been validated through detailed experimental evaluations, which demonstrate its effectiveness in balancing training objectives and improving generalization. Specific experimental results and analysis will be detailed in Section 3.2.

2.3 NOMS-FF Model Architecture

The NOMS-FF model is designed to address the challenges faced by traditional LDPC decoders, particularly under varying noise conditions. By leveraging the iterative message-passing structure of Tanner graphs and incorporating trainable normalization factors, offset parameters, and a relaxation mechanism, the model provides increased flexibility and robustness while maintaining computational efficiency.

In the NOMS-FF model, decoding involves two types of messages: **VC messages** and **CV messages**. These messages are exchanged iteratively between VNs and CNs along the edges of the Tanner graph. VC messages pass information from VN to CN, while CV messages represent the feedback from CN to VN. This iterative exchange refines the decoding process and adapts to the noise characteristics of the channel.

The model also incorporates a relaxation mechanism inspired by factor graph message-passing principles [12], where the relaxation factor R balances the influence of consecutive iterations, smoothing updates and stabilizing convergence characteristics commonly analyzed via density evolution [7].

The decoding process starts with the channel-received sequence $\mathbf{y}$, obtained after BPSK modulation and transmission through an AWGN channel. The decoding procedure is detailed in Algorithm 1.

Algorithm 1 Decoding Process in the NOMS-FF Model

Require: Received sequence $\mathbf{y}$, parity-check matrix $\mathbf{H}$, number of iterations L , relaxation factor R .

Ensure: Decoded codeword $\hat{\mathbf{c}}$.

Initialization:

Initialize VC messages: $\mathbf{vc}_{i \rightarrow j} = \mathbf{y}_i, \forall (i, j)$.

Initialize CV messages: $\mathbf{cv}_{j \rightarrow i} = 0, \forall (j, i)$.

Initialize relaxed messages: $\mathbf{M}_t = \mathbf{vc}$.

for $l = 1$ **to** L **do**

Step 1: Compute VC messages (Variable-to-Check).

$$\mathbf{vc}_{i \rightarrow j}^{(l)} = \mathbf{y}_i + \sum_{k \in \mathcal{N}(i) \setminus j} \mathbf{cv}_{k \rightarrow i}^{(l-1)}.$$

Step 2: Relaxation Update

$$M_t^{(l)} = R \cdot M_t^{(l-1)} + (1 - R) \cdot \mathbf{vc}_{i \rightarrow j}^{(l)}.$$

Step 3: Compute CV messages (Check-to-Variable)

$$\mathbf{cv}_{j \rightarrow i}^{(l)} = \prod_{k \in \mathcal{N}(j) \setminus i} \text{sign}(M_t^{(l)}(k \rightarrow j)) \cdot a_{kj} \cdot \max\left(\min_{k \in \mathcal{N}(j) \setminus i} |M_t^{(l)}(k \rightarrow j)| - b_{kj}, 0\right).$$

where a_{kj} and b_{kj} are the trainable normalization (scaling) and offset parameters, respectively.

Step 4: Marginalization (Soft Decision Calculation)

$$\hat{\mathbf{c}}_i^{(l)} = \mathbf{y}_i + \sum_{j \in \mathcal{N}(i)} \mathbf{cv}_{j \rightarrow i}^{(l)}.$$

end for

Hard Decision:

Apply thresholding to generate the final decoded code-word: $\hat{\mathbf{c}} = \text{sign}(\hat{\mathbf{c}})$.

By incorporating trainable normalization parameters a_{kj} and offset parameters b_{kj} , the NOMS-FF model adaptively adjusts message magnitudes during the decoding process, assisting performance under varying noise conditions. The relaxation mechanism further ensures stable updates across iterations, accelerating convergence and supporting robustness.

The iterative process concludes with a soft decision vector $\hat{\mathbf{c}}$, which is transformed into a binary sequence through a final thresholding step. Compared to traditional LDPC decoders, the NOMS-FF model provides adaptability in diverse noise environments, as evaluated in our experiments.

2.4 Layer-Wise Post-Training Framework

To support the model's reliability and efficiency during post-training, this paper utilizes a layer-wise post-training framework. It consists of two complementary strategies: **layer-freezing training**, which stabilizes the learning process by selectively locking parameters of certain layers, and **layer-wise dynamic SNR adjustment**, which aligns training with the noise characteristics of each layer. These strategies collectively support the generalization ability of the model and mitigate overfitting.

Layer-Freezing Training

Layer freezing is a training technique designed to stabilize the learning process by locking the parameters of specific layers during training. As demonstrated in works on greedy layer-wise training [2], isolating layers during optimization can yield localized

representations, enabling fine-tuning of unfrozen layers while maintaining the stability of frozen ones. Building upon this, and referring to curriculum learning strategies [3], structuring training stages progressively can assist optimization.

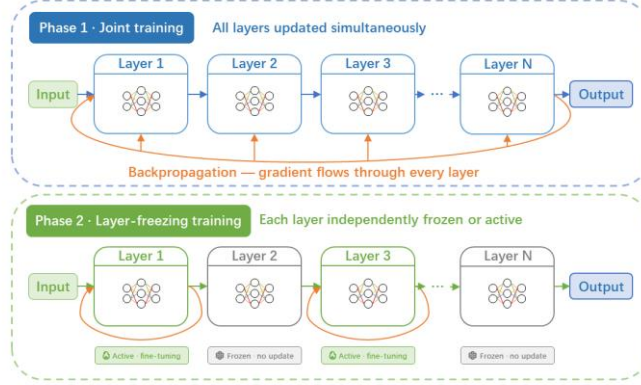


Fig. 2. Joint training and the layer-freezing training in the NOMS-FF decoder.

As shown in **Fig. 2**, layer freezing in this study is applied to the decoder's deep learning components, particularly in the computation of CV messages. The NOMS algorithm optimizes these computations, where the deep learning component includes the trainable normalization matrix a and offset matrix b , as described in Algorithm 1. During training, frozen layers participate in forward propagation to compute updated messages, but their parameters remain fixed and are excluded from gradient updates during backpropagation. In contrast, unfrozen layers undergo regular fine-tuning to adapt to the training data.

To further aid the model's generalization, the third term in the loss function (Eq. 8) is introduced as a regularization term. This term addresses overfitting and stabilizes the learning process when applying the layer-freezing technique.

Layer-Wise Dynamic SNR Adjustment

The iterative decoding nature of BP decoders results in different layers processing varying noise levels during pre-training. Consequently, each layer processes specific SNR ranges. To utilize these layer-specific characteristics during post-training, this paper proposes a layer-wise dynamic SNR adjustment strategy as follows:

1. **SNR Range Definition:** The SNR for each layer is defined as:

$$\text{SNR}_i = \text{SNR}_{\min} + (\text{SNR}_{\max} - \text{SNR}_{\min}) \cdot (1 - \alpha^i), \quad (12)$$

where $\text{SNR}_{\min}$ and $\text{SNR}_{\max}$ represent the minimum and maximum SNR values, α is the exponential decay factor, and i denotes the layer index. This ensures that higher layers are assigned progressively lower SNR values, simulating noisier conditions.

2. **Noise Standard Deviation Calculation:** The noise standard deviation σ for each layer is calculated based on the SNR value:

$$\sigma = \sqrt{\frac{1}{2 \cdot \frac{k}{n} \cdot 10^{\frac{\text{SNR}}{10}}}} \quad (13)$$

where k and n are the number of message bits and the codeword length, respectively, and SNR is the layer-specific SNR. The computed σ determines the intensity of noise added to the data.

3. **Batch Size Distribution:** The training batch size is distributed across layers using a weighted strategy:

$$w_i = \text{BatchSize}_{\text{total}} \cdot \alpha^i, \quad (14)$$

$$\text{BatchSize}_i = \frac{w_i}{\sum_j w_j} \cdot \text{BatchSize}_{\text{total}}, \quad (15)$$

where w_i is the weight for layer i , BatchSize_i is the batch size allocated to layer i , and $\text{BatchSize}_{\text{total}}$ is the total batch size. This ensures that layers with lower SNRs (higher noise levels) receive proportionally larger training data.

3 Experimental Results

The proposed model is implemented using the PyTorch framework, with the Adam optimizer [10] for training. The batch size is set to 288, and the pre-training process consists of 2000 steps. For LDPC decoding, both traditional and deep learning-based models use 50 iterations. This section details the performance of the model during pre-training and post-training, analyzing its effectiveness from a layer-wise perspective.

All the tests in this section are conducted on standard QC-LDPC codes, which serve as a baseline for 6G research [18]. To comprehensively evaluate the generalization capability of the models, we adopt three distinct configurations with a design code rate of $R = 0.33$: Base Graph 2 (BG2) with a code length of $N = 240$, $K = 80$ and Base Graph 1 (BG1) with code lengths of $N = 2016$, $K = 672$ and $N = 4032$, $K = 1344$.

3.1 Performance of the NOMS-FF Model

Fig. 3. Training Loss vs. Epochs (Smoothed Curve) shows the loss curve of the NOMS-FF model during training, where the loss is calculated using Equation (6). From the figure, it can be observed that the training curve decreases steadily during the first 200 epochs and stabilizes around epoch 300, with the loss converging to approximately 0.12. Afterward, the loss remains relatively consistent throughout the rest of the training process.

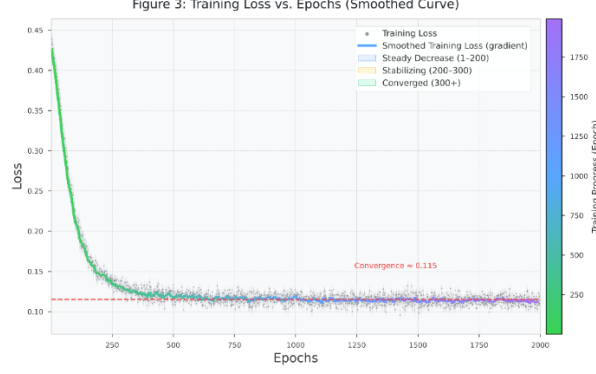


Fig. 3. Training Loss vs. Epochs (Smoothed Curve).

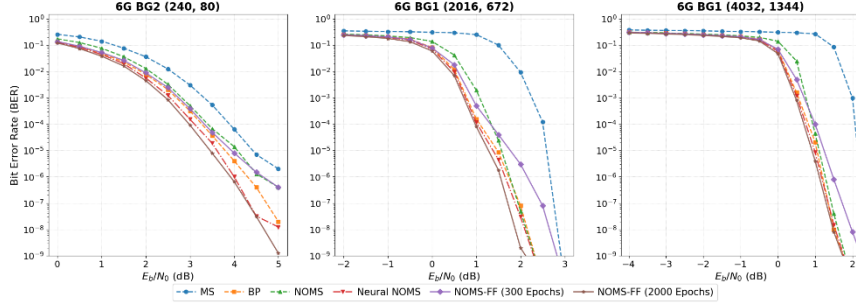


Fig. 4. Performance Comparison of NOMS-FF and Baseline Decoders Across Multiple Code Lengths.

To evaluate the model's performance, compare the BER of the NOMS-FF model with traditional algorithms (MS, NOMS, BP) and a baseline deep learning decoder (Neural NOMS) across the SNR range of $[0, 5]$ dB. The results include the NOMS-FF model trained for 300 epochs and 2000 epochs. The observations are as follows:

To evaluate the model's performance and generalization, **Fig. 4** compares the BER of the NOMS-FF model with traditional algorithms (MS, NOMS, BP) and a state-of-the-art deep learning baseline decoder (Neural NOMS) across the three different code lengths and their corresponding SNR ranges. Notably, the Neural NOMS baseline specifically adopts the advanced Type-V architecture proposed in [8], serving as a highly competitive benchmark. The observations for the 300-epoch and 2000-epoch models are consistent across these varying conditions. In the high-noise regions of all evaluated code lengths, the 300 epochs model effectively mitigates errors and remains competitive. However, it consistently experiences an early error floor in the high SNR ranges across all tested lengths, eventually underperforming compared to traditional BP. Conversely, the 2000 epochs model overcomes local fitting through extended training with the syndrome loss. It achieves a steeper slope in the waterfall region and completely eliminates the error floor across all tested lengths, demonstrating robust generalization

from short codes $N = 240$ to long codes $N = 4032$ across their respective SNR operating ranges.

Analyzing **Fig. 3** and **Fig. 4**, it is observed that while the training loss exhibits minimal differences around 300 epochs and 2000 epochs, the corresponding performance variations are clear. According to standard deep learning training patterns, a stable loss at around 300 epochs typically indicates sufficient training, and further training to 2000 epochs can lead to overfitting. However, for the NOMS-FF model, the 2000 epochs model provides better results than the 300 epochs model during testing.

This outcome is attributed to the inclusion of the Syndrome Loss mechanism, which mitigates overfitting and allows the model to utilize larger datasets for extended training. Without the Syndrome Loss, the model exhibits difficulties in achieving convergence despite hyperparameter tuning and training process adjustments. This indicates the role of the Syndrome Loss in the proposed architecture's training phase.

3.2 Results of Layer-Wise Post-Processing Training

This section presents the post-training and layer-wise evaluation results based on the NOMS-FF model trained for 2000 epochs. To analyze the impact of different layers, independent layer-specific experiments were conducted.

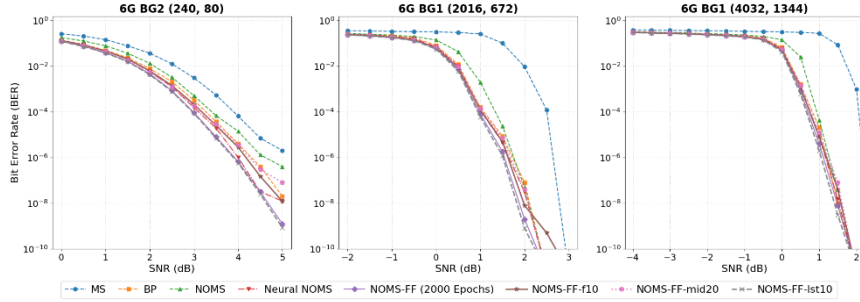


Fig. 5. Performance Comparison of Partial-Layer Decoding Models Across Multiple Code Lengths.

Fig. 5 shows the BER results of these experiments evaluated across the three distinct code lengths. All models were derived from the corresponding NOMS-FF (2000 Epochs) models. Specifically, f10, mid20, and lst10 represent models where only the first 10 iterations, the middle 20 iterations, and the last 10 iterations, respectively, were executed using the deep learning model, while the remaining iterations were performed using the traditional BP decoder.

The results indicate that deploying deep learning modules exclusively in specific stages of the decoding process yields distinct behaviors consistent across varying lengths and SNRs. The f10 model demonstrates robust initial noise suppression but exhibits a slight error floor at extremely high SNRs. The mid20 model accelerates intermediate convergence but degrades toward traditional BP performance in the deep error floor region. Conversely, the lst10 model emerges as the optimal configuration.

By allowing the traditional BP to handle the global linear message passing in the early stages and utilizing the neural network to precisely mitigate trapping sets in the final iterations, lst10 not only maintains steep waterfall characteristics but also suppresses the error floor entirely. It consistently matches or marginally outperforms the full-layer NOMS-FF (2000 Epochs) model across the high SNR regions of all three code lengths.

Moreover, for models where only partial layers are involved in computation (e.g., lst10), the forward-pass computational complexity is substantially reduced compared to the full NOMS-FF model, achieving inference speeds comparable to traditional BP decoders while delivering superior accuracy.

For subsequent fine-tuning, a combined strategy of Layer-wise Fine-Tuning and Freezing Upper Layers was adopted. Specifically, the model was first fine-tuned layer by layer, followed by freezing the upper layers to further optimize the lower layers. During the entire fine-tuning process, the loss function defined in Equation (8) was used.

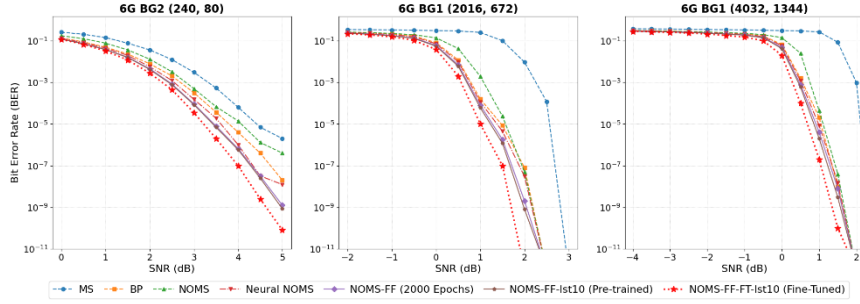


Fig. 6. Performance Comparison of Fine-Tuned Models Across Multiple Code Lengths.

As shown in **Fig. 6**, the NOMS-FF-FT-lst10 (Fine-Tuned) model refers to a fine-tuned version of the pre-trained lst10 model, evaluated across the three code lengths. In this configuration, the traditional BP iterations were frozen, and the remaining 10 neural network layers were subjected to a targeted global fine-tuning using the soft-syn-drome loss. The results show that the fine-tuned model establishes an absolute advantage over all other baselines regardless of code length or specific SNR range. Notably, in the high SNR deep-water regions corresponding to each code, the fine-tuning process specifically optimized the neural weights to overcome the unique residual errors of the BP algorithm. The FT-lst10 model plunges to unprecedented BER levels, achieving full order-of-magnitude improvements over the pre-trained lst10 models. This profound error floor suppression scales robustly to the longer $N=2016$ and $N=4032$ codes. Furthermore, as only the last 10 layers are involved in deep learning decoding, this ultimate configuration provides an optimal trade-off: state-of-the-art decoding performance with minimal computational overhead across varying code lengths. While NOMS-FF achieves superior performance across all tested code lengths, it is essential to understand the individual contribution of each proposed component. Thus, a comprehensive ablation study is conducted in the following section.

3.3 Ablation Study and Mechanism Analysis

To rigorously evaluate the individual contributions of the proposed mechanisms and justify the introduced architectural complexity, an extensive ablation study was conducted across multiple 6G-candidate QC-LDPC code lengths ($N = 240$, $N = 2016$, and $N = 4032$). We defined four model configurations by progressively incorporating our proposed strategies, with their BER performances at the critical deep-water SNRs summarized in **Table 2**.

- **Model A (Baseline Neural NOMS):** The baseline deep learning decoder, specifically adopting the state-of-the-art Type-V architecture proposed in [8]. It is trained end-to-end using solely the standard Cross-Entropy (CE) loss, without layer-wise freezing or dynamic SNR allocation.
- **Model B (+ Soft-Syndrome Loss):** Builds upon Model A by integrating the proposed soft-syndrome loss function ($\mathcal{L}_{syn}$) into the training phase, structurally constraining the network to learn parity-check consistency.
- **Model C (+ Layer-Freezing):** Builds upon Model B by implementing the layer-freezing post-training strategy, locking the early layers to preserve linear message-passing features while fine-tuning the upper layers.
- **Model D / Final (+ Dynamic SNR Allocation):** The ultimate configuration (equivalent to the NOMS-FF-FT-lst10), which incorporates the layer-wise dynamic SNR adjustment strategy on top of Model C, enabling targeted deep-layer optimization.

Table 1. Configuration of models in the ablation study.

Configurations	$\mathcal{L}_{syn}^1$	LF ²	DSA ³
Model A: Neural NOMS (Type-V) [8]	—	—	—
Model B: NOMS-FF (Initial)	✓	—	—
Model C: NOMS-FF + LF	✓	✓	—
Model D (Final): FT-lst10	✓	✓	✓

Table 2. Ablation results across multiple QC-LDPC code lengths.

Configurations	BG2 (240) @ 5.0dB	BG1 (2016) @ 2.0dB	BG1 (4032) @ 1.5dB
Model A: Neural NOMS (Type-V) [8]	1.23×10^{-8}	3.12×10^{-8}	1.58×10^{-8}
Model B: NOMS-FF (Initial)	1.25×10^{-9}	2.08×10^{-9}	8.45×10^{-9}
Model C: NOMS-FF + LF	8.50×10^{-10}	8.15×10^{-10}	3.24×10^{-9}
Model D (Final): FT-lst10	8.00×10^{-11}	1.15×10^{-12}	1.06×10^{-10}

As observed in **Table 1** and **Table 2**, **Model A**, despite utilizing the sophisticated Type-V neural architecture, encounters a bottleneck in the deep-water region across all

¹ $\mathcal{L}_{syn}$: Soft-Syndrome Loss.

² LF: Layer-Freezing.

³ DSA: Dynamic SNR Allocation.

code lengths. It yields early error floors, such as 1.23×10^{-8} for the short code and 1.58×10^{-8} for the long code. This indicates that standard CE loss alone is insufficient to globally resolve deep trapping sets, regardless of the codeword dimension.

The introduction of the soft-syndrome loss in **Model B** yields a non-trivial breakthrough, effectively breaking the error floor and improving the BER by roughly an order of magnitude across the board (e.g., dropping to 1.25×10^{-9} at 5.0 dB for $N = 240$). This substantial gain confirms that the soft-syndrome loss is a fundamental structural enhancement rather than a marginal modification, exhibiting excellent generalization from short to long codes.

Furthermore, **Model C** demonstrates that layer-freezing provides enhanced stability during long-term training, preventing parameter drift and slightly pushing the performance further down, verifying its utility in isolating the linear and non-linear message-passing domains.

Finally, **Model D** establishes the absolute advantage of the full framework. By matching the deep layers with dynamically allocated low-noise data distributions, the DSA mechanism aligns the training distribution with the actual noise floor of iterative decoding. Our ultimate **FT-1st10** configuration, which employs targeted global fine-tuning on top of the proposed framework, pushes the BER to unprecedented levels (e.g., 1.06×10^{-10} for $N = 4032$ and nearly 10^{-12} for $N = 2016$). This progressive enhancement rigorously validates that each architectural component contributes substantially to the overall decoding robustness across varying code lengths.

3.4 Complexity Analysis

To evaluate the computational efficiency of the proposed NOMS-FF decoder, we analyze the operations required for one decoding iteration and compare it with traditional iterative decoding algorithms, including SP, MS, NMS, and OMS. Following the analysis methodology presented in [8], **Table 3** summarizes the number of basic mathematical operations required, where E denotes the total number of edges in the Tanner graph.

Table 3. Operations Required for One Iteration.

Operation	$\tanh$	Multiplications ($\times$)	Additions (+)	Comparisons	Sign flips
SP	$2E$	$\approx 2E$	$\approx 2E$	—	—
MS	—	—	$\approx 2E$	$\approx 2E$	$\approx 2E$
NMS	—	E	$\approx 2E$	$\approx 2E$	$\approx 2E$
OMS	—	—	$\approx 3E$	$\approx 2E$	$\approx 2E$
NOMS-FF	—	$\approx 1E$	$\approx 3E$	$\approx 2E$	$\approx 2E$
NOMS	—	$\approx 1E$	$\approx 3E$	$\approx 2E$	$\approx 2E$
Neural-NOMS	—	$\approx 2E$	$\approx 4E$	$\approx 2E$	$\approx 2E$

As shown in **Table 3**, the NOMS-FF model avoids the computationally intensive hyperbolic tangent ($\tanh$) operations required by the standard SP algorithm [8]. Notably, the computational complexity of the proposed NOMS-FF model is exactly identical to that of the standard NOMS algorithm. Furthermore, compared to the state-of-the-art Neural NOMS (Type-V) baseline, the NOMS-FF framework exhibits a slightly lower computational overhead per iteration. While the introduction of learnable scaling factors, offset parameters, and the relaxation mechanism requires additional multiplications and additions compared to the standard MS and OMS algorithms, the overall computational complexity remains strictly linear with respect to the number of edges E . Moreover, our partial-layer decoding strategy allows the network to freeze or bypass specific computations in certain layers, providing a flexible trade-off between decoding accuracy and hardware inference speed without fundamentally increasing the asymptotic complexity.

References

1. Artemasov, D., Andreev, K., Rybin, P., Frolov, A.: Soft-output deep neural network-based decoding. In: 2023 IEEE Globecom Workshops (GC Wkshps). pp. 1692–1697. IEEE (2023). <https://doi.org/10.1109/GCWkshps58843.2023.10464942>
2. Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H.: Greedy layer-wise training of deep networks. In: Advances in Neural Information Processing Systems (NeurIPS). vol. 19, pp. 153–160 (2007)
3. Bengio, Y., Louradour, J., Collobert, R., Weston, J.: Curriculum learning. In: Proceedings of the 26th Annual International Conference on Machine Learning (ICML). pp. 41–48. ACM (2009). <https://doi.org/10.1145/1553374.1553380>
4. Bennatan, A., Choukroun, Y., Kisilev, P.: Deep learning for decoding of linear codes - a syndrome-based approach. In: 2018 IEEE International Symposium on Information Theory (ISIT). pp. 1595–1599. IEEE (2018). <https://doi.org/10.1109/ISIT.2018.8437340>
5. Bruck, J., Blaum, M.: Neural networks, error-correcting codes, and polynomials over the binary n -cube. IEEE Trans. Inf. Theory 35, 976–987 (1989). <https://doi.org/10.1109/18.42215>
6. Chen, J., Dholakia, A., Eleftheriou, E., Fossorier, M.P.C., Hu, X.Y.: Reduced-complexity decoding of LDPC codes. IEEE Trans. Commun. 53, 1288–1299 (2005). <https://doi.org/10.1109/TCOMM.2005.852852>
7. Chen, J., Fossorier, M.P.C.: Density evolution for two improved BP-based decoding algorithms of LDPC codes. IEEE Commun. Lett. 6, 208–210 (2002). <https://doi.org/10.1109/4234.1001666>
8. Dai, J., Tan, K., Si, Z., Niu, K., Chen, M., Poor, H.V., Cui, S.: Learning to decode protograph LDPC codes. IEEE J. Sel. Areas Commun. 39, 1983–1999 (2021). <https://doi.org/10.1109/JSAC.2021.3078488>
9. Fossorier, M.P.C., Mihaljevic, M., Imai, H.: Reduced complexity iterative decoding of low-density parity check codes based on belief propagation. IEEE Trans. Commun. 47, 673–680 (1999). <https://doi.org/10.1109/26.768759>
10. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016)
11. Gruber, T., Cammerer, S., Hoydis, J., ten Brink, S.: On deep learning-based channel decoding. In: 2017 51st Annual Conference on Information Sciences and Systems (CISS). pp. 1–6. IEEE (2017). <https://doi.org/10.1109/CISS.2017.7926071>

12. Kschischang, F.R., Frey, B.J., Loeliger, H.A.: Factor graphs and the sum-product algorithm. *IEEE Trans. Inf. Theory* 47, 498–519 (2001). <https://doi.org/10.1109/18.910572>
13. Liao, S., Deng, C., Yin, M., Yuan, B.: Doubly residual neural decoder: Towards low-complexity high-performance channel decoding. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. vol. 35, pp. 8574–8582. AAAI Press (2021). <https://doi.org/10.1609/aaai.v35i10.17040>
14. Lugosch, L., Gross, W.J.: Learning from the syndrome: Decoding of polar codes using neural networks. In: *2018 52nd Asilomar Conference on Signals, Systems, and Computers*. pp. 594–598. IEEE (2018). <https://doi.org/10.1109/ACSSC.2018.8645229>
15. Nachmani, E., Be'ery, Y., Burshtein, D.: Learning to decode linear codes using deep learning. In: *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. pp. 341–346. IEEE (2016). <https://doi.org/10.1109/ALLERTON.2016.7852251>
16. Nachmani, E., Marciano, E., Lugosch, L., Gross, W.J., Burshtein, D., Be'ery, Y.: Deep learning methods for improved decoding of linear codes. *IEEE J. Sel. Top. Signal Process.* 12, 119–131 (2018). <https://doi.org/10.1109/JSTSP.2017.2788405>
17. O'Shea, T.J., Hoydis, J.: An introduction to deep learning for the physical layer. *IEEE Trans. Cogn. Commun. Netw.* 3, 563–575 (2017). <https://doi.org/10.1109/TCCN.2017.2758370>
18. Rowshan, M., Qiu, M., Xie, Y., Gu, X., Yuan, J.: Channel coding toward 6G: Technical overview and outlook. *IEEE Open J. Commun. Soc.* 5, 2585–2685 (2024). <https://doi.org/10.1109/OJCOMS.2024.3390000>
19. Shafi, M., Molisch, A.F., Smith, P.J., Haustein, T., Murphy, P., Silva, I.D., Tuomaala, E., Palleh, A.: 5G: A tutorial overview of standards, trials, challenges, deployment, and practice. *IEEE J. Sel. Areas Commun.* 35, 1201–1221 (2017). <https://doi.org/10.1109/JSAC.2017.2692307>
20. Teich, J.M., Means, R.E.: Neural network decoding of convolutional codes. *IEEE J. Sel. Areas Commun.* 8, 1691–1699 (1990)
21. Teng, C.F., Wu, C.H., Ho, A.K., Wu, A.Y.: Low-complexity neural network decoders for polar codes. *IEEE Trans. Commun.* 68, 3325–3335 (2020). <https://doi.org/10.1109/TCOMM.2020.2982833>
22. Shah, N., Vasavada, Y.: Neural layered decoding of 5G LDPC codes. *IEEE Commun. Lett.* 25, 3590–3593 (2021). <https://doi.org/10.1109/LCOMM.2021.3113610>