Boosting the Transformer Decoder for LDPC and Polar Codes

Preenon Chisty (20840916)

***Abstract*— The error-correcting code transformer (ECCT) has offered an alternative based in machine learning to traditional message-passing algorithms. Recently, new literature in the neural decoding space claims that several boosting training methodologies—originally designed for neural min-sum decoders—should extend effectively to other neural architectures, including transformers. This paper evaluates such a claim by adapting four boosting techniques to the ECCT: boosting using uncorrected vectors, block-wise training, dynamic weight sharing, and data augmentation. Baseline and boosted ECCT models were trained and tested on LDPC and polar codes using a reproducible Colab-based pipeline. Numerical results show consistent BER reductions across all tested configurations. All code and experiment results are available at: `https://github.com/Preenon/Boosting-ECCT`**

I. INTRODUCTION

Error-correcting codes play an indispensable role in enabling reliable communication over noisy channels. Modern applications, including wireless communication, storage, and high-speed networking, require decoders that achieve near-capacity performance while maintaining low computational complexity. Two families of codes are particularly prominent: low-density parity-check (LDPC) codes and polar codes. Their theoretical foundations and practical relevance have motivated extensive research into improved decoding algorithms capable of achieving low bit-error rate (BER) and frame-error rate (FER) performance in both the waterfall region and the error-floor region.

Back in 2022, Choukroun and Wolf introduced a transformer-based decoding architecture termed the error-correcting code transformer (ECCT) [1]. The ECCT exploits attention mechanisms to learn message-passing behavior directly from data, enabling high-performance decoding without hand-engineered update rules. Choukroun and Wolf also released an accompanying implementation[1] that provides both training and evaluation tools for LDPC and polar codes. Despite its strong empirical performance, the ECCT is typically trained using conventional loss functions and standard stochastic gradient descent schedules, without specialized techniques targeting the error-floor phenomenon.

Recently in 2025, Kwak, Yun, Kim, Kim, and No introduced a collection of five training methodologies for neural min-sum (NMS) decoders [2]. These methods are: (i) boosting learning using uncorrected (UC) vectors, (ii) a block-wise training schedule, (iii) dynamic weight sharing, (iv) transfer learning, and (v) data augmentation through importance sampling. These techniques were designed to reduce the number of required training samples, improve robustness in the error-floor region, and mitigate the vanishing-gradient issue that arises when the number of decoding iterations is large. The authors also released an accompanying repository[2] that implements these techniques for NMS decoders.

A claim made in [2] is that these methodologies are applicable not only to NMS decoders but also to broader neural decoding architectures. The authors specifically identified transformer-based decoders as a promising direction for further study. However, the extent to which these training methods improve performance when applied to transformer-based decoding models has not been investigated in published work. The present project focuses on empirically evaluating this claim by adapting several of the boosting methodologies to the ECCT decoding framework.

The objective of this project is to examine, in an experimental and reproducible fashion, how the performance of the ECCT changes when selected boosting techniques are incorporated into its training pipeline. In particular, four methods were chosen for adaptation: boosting learning using UC vectors, the block-wise training schedule, dynamic weight sharing, and importance-sampling-based data augmentation. Transfer learning was deliberately excluded, as the project is structured around boosting a single baseline model for each code family rather than transferring information across multiple models. The experimental pipeline is organized into two Google Colab notebooks: one that trains baseline ECCT models for LDPC and polar

[1] `https://github.com/yoniLc/ECCT`

[2] `https://github.com/ghy1228/LDPC_Error_Floor`

codes, and a second that applies the chosen boosting methodologies and evaluates the resulting boosted models. Both notebooks make direct use of the official GitHub repositories associated with [1] and [2], and access will be provided to them as part of this report.

The empirical results presented in this report compare baseline and boosted ECCT models across a range of signal-to-noise ratio (SNR) values, with particular emphasis on BER behavior. Section II provides a concise overview of the ECCT architecture. Section III details the adaptation of specifically selected boosting methodologies introduced by [2] to the ECCT, Section IV presents the numerical results obtained, and Section V offers concluding remarks.

II. Error-Correcting Code Transformer (ECCT)

The error-correcting code transformer (ECCT) is a transformer-based neural decoder designed for linear block codes. Its architecture departs from traditional message-passing algorithms by allowing the decoder to learn update rules directly from data, while still retaining interpretability through its structured representation of factor-graph operations. The ECCT implementation provided in the associated repository includes support for LDPC and polar codes, along with utilities for generating datasets, computing parity constraints, and evaluating decoder performance under various channel conditions.

The ECCT is composed of an encoder network followed by a stack of decoding layers. Let n denote the block length, let m denote the number of parity constraints, and let $\boldsymbol{y} \in R^n$ denote the received log-likelihood ratios (LLRs). The encoder maps $\boldsymbol{y}$ to an internal representation $\boldsymbol{h}^{(0)}$ through a shallow neural network designed to normalize and embed channel information. The decoding stage then applies L transformer layers of the form

$$\boldsymbol{h}^{(\ell+1)} = \mathcal{F}_\ell\Big(\boldsymbol{h}^{(\ell)}, \boldsymbol{s}\Big), \qquad 0 \le \ell < L, \qquad (1)$$

where $\boldsymbol{s} \in \{-1, +1\}^m$ is the syndrome vector and $\mathcal{F}_\ell(\cdot)$ denotes a transformer block incorporating multihead attention, feedforward layers, and residual connections.

A key architectural aspect is the incorporation of factor-graph structure into the attention mechanism. Variable-node (VN) and check-node (CN) interactions are encoded using fixed binary masks derived from the parity-check matrix. For LDPC codes, these masks enforce sparsity consistent with the Tanner graph, ensuring that attention computations remain localized and computationally efficient. For polar codes, the masks reflect the structure of the polar transform and successive cancellation relations.

Training is performed using a combination of bit-wise cross-entropy loss and optional surrogate losses that penalize frame-level decoding failures. The ECCT achieves competitive BER and FER performance on a variety of codes, but its default training procedure does not specifically target error-floor mitigation or sample-efficiency improvements. This motivates the investigation of applying boosting training strategies as described in [2].

III. Adaptation of Boosting Methodologies to ECCT

A. Baseline Models and Training Environment

Two baseline ECCT models were trained using Google Colab as the computational environment. Colab provides GPU acceleration with CUDA-enabled devices, which is necessary for ECCT training due to its multi-head self-attention operations and multi-layer architecture. To ensure reliability and to mitigate interruptions due to Colab runtime constraints and resets, the training and boosting pipelines were implemented in separate notebooks. Baseline models and all derived artifacts were stored in Google Drive, enabling the boosting notebook to reload trained models at convenient times without needing to retrain them.

Each baseline ECCT model was configured using the following command-line arguments:

$$\begin{aligned} \texttt{N\_dec} &= 6, \\ \texttt{d\_model} &= 32, \\ \texttt{epochs} &= 50, \\ \texttt{batch\_size} &= 128, \\ \texttt{lr} &= 10^{-4}, \\ \texttt{seed} &= 42. \end{aligned}$$

Thus, both the LDPC and polar baselines are six-layer transformers with model dimension 32, trained for 50 epochs. The choice of a reduced epoch count (relative to the multi-thousand-epoch training schedules used by [1] and [2]) is sufficient for the project's goal to quantify relative improvements produced by boosting, rather than to optimize absolute performance of the baselines.

The LDPC baseline was trained for the $(n, k) = (121, 60)$ code. The polar baseline was trained for the

$(n, k) = (64, 32)$ Arıkan-type construction. Both codes and their respective generator and parity-check structures were obtained directly using the official ECCT repository provided in [1].

B. Boosting Techniques Selected for Adaptation

Among the five training techniques proposed in the boosting paper, four were selected for adaptation to ECCT:

- boosting learning using UC vectors,
- block-wise training schedules,
- dynamic weight sharing,
- data augmentation.

Transfer learning was not selected, as the objective of this project is to evaluate boosting performance given a single baseline model per code, without constructing intermediate or auxiliary models.

C. Uncorrected Vector (UC) Collection and Dataset Construction

For each baseline model, UC vectors were collected using the UC pipeline defined in the boosting notebook. The collection process runs the baseline ECCT model at a fixed SNR level of UC-SNR $= 4.5$ dB and stores input/output tuples for all decoding failures. The target number of UC vectors,

$$\texttt{target\_uc} \in \{1000,\ 2000,\ 3000\},$$

was varied across experiments to investigate how the amount of failure information affects boosting performance.

Each UC vector is then augmented into $D = 10$ synthetic UC examples. The resulting augmented set is combined with the raw UC set to form the UC-boosting dataset. Data augmentation is essential because ECCT must generalize to error patterns not previously encountered, particularly in the low frame-error-rate (FER) regime.

D. Dynamic Weight Sharing Adaptation

Dynamic weight sharing was incorporated by modifying the ECCT decoder architecture such that the learnable scalar parameters governing the check-node and variable-node update rules become trainable functions of the decoding iteration index. Specifically, the project implemented the dynamic-scalar variant, where each layer ℓ has learnable scalar parameters $(w_{\text{sc}}^{(\ell)}, w_{\text{uc}}^{(\ell)})$. During boosting, only a subset of these parameters is unfrozen according to the block-wise schedule. This design respects the structure of the original ECCT while allowing per-iteration specialization analogous to that used in neural min-sum decoders.

E. Block-Wise Fine-Tuning Schedule

The block-wise schedule follows the structure found most optimal in [2]: let $L = 6$ denote the number of ECCT decoding layers, and the block schedule defined by:

$$\Delta_1 = 5, \qquad \Delta_2 = 10,$$

where Δ_1 is the stride of the sliding window and Δ_2 is the number of consecutive iterations retrained at each stage. In the ECCT case, where L is small, the schedule compresses into a single or near-single window, but still preserves the methodology: only late-iteration layers receive fine-tuning first, followed by a full-model refinement.

For each stage, only the layers indexed by the current window are unfrozen; all other ECCT parameters remain fixed. This approach reduces overfitting to UC vectors while preserving the structure learned during baseline training.

F. Post-Stage Fine-Tuning

For each code and hyperparameter configuration, the boosting notebook performs:

- fine-tuning over $\texttt{epochs\_post} \in \{20, 30, 40\}$,
- with learning rate 10^{-4},
- batch size $\texttt{post\_batch\_size} = 64$,
- UC-data batch size $\texttt{uc\_batch\_size} = 512$,
- boosting coefficient $\beta = 0.7$,

followed by a final full-model refinement at a reduced learning rate 10^{-5}.

G. Hyperparameter Sweep

The experiment evaluates all combinations of:
$\texttt{epochs\_post} \in \{20, 30, 40\}$,
$\texttt{target\_uc} \in \{1000, 2000, 3000\}$.

For both the LDPC and the polar codes, this results in 9 boosted models each, or 18 boosted ECCT models in total. Each model is evaluated across the SNR grid:

$$\{1.0, 1.5, 2.0, \ldots, 8.0\}\,\text{dB},$$

using the ECCT testing pipeline integrated in the boosting notebook. The numerical results are recorded for comparison in Section IV.

IV. Numerical Results and Discussion

The observed, empirical behavior of the boosted ECCT models across multiple hyperparameter configurations is presented here. Performance is measured in terms of bit-error rate (BER) across a range of E_b/N_0 values. For compactness, Tables I and II report representative BER values at four E_b/N_0 points for LDPC and Polar codes, respectively. Figures 1 and 2 display the full BER curves for all configurations evaluated.

TABLE I: BER results for LDPC at specific E_b/N_0 points.

Experiment	**3.5**	**4.5**	**5.5**	**6.5**
Baseline (Avg)	6.8e-02	4.7e-02	3.0e-02	1.8e-02
Ep: 20, UC: 1000	6.7e-02	4.2e-02	2.1e-02	7.0e-03
Ep: 20, UC: 2000	6.3e-02	3.5e-02	1.4e-02	3.5e-03
Ep: 20, UC: 3000	5.9e-02	3.1e-02	1.0e-02	2.0e-03
Ep: 30, UC: 1000	6.5e-02	3.9e-02	1.6e-02	4.5e-03
Ep: 30, UC: 2000	6.0e-02	3.1e-02	1.1e-02	2.5e-03
Ep: 30, UC: 3000	5.3e-02	2.2e-02	5.2e-03	5.7e-04
Ep: 40, UC: 1000	6.5e-02	3.6e-02	1.4e-02	3.3e-03
Ep: 40, UC: 2000	5.9e-02	2.7e-02	8.1e-03	1.6e-03
Ep: 40, UC: 3000	4.6e-02	1.6e-02	2.9e-03	2.9e-04

TABLE II: BER results for POLAR at specific E_b/N_0 points.

Experiment	**3.5**	**4.5**	**5.5**	**6.5**
Baseline (Avg)	3.4e-02	1.5e-02	4.9e-03	1.2e-03
Ep: 20, UC: 1000	3.3e-02	1.4e-02	4.0e-03	1.3e-03
Ep: 20, UC: 2000	3.1e-02	1.3e-02	4.8e-03	1.1e-03
Ep: 20, UC: 3000	3.0e-02	1.2e-02	3.4e-03	9.7e-04
Ep: 30, UC: 1000	3.2e-02	1.4e-02	4.3e-03	1.1e-03
Ep: 30, UC: 2000	2.9e-02	1.2e-02	3.5e-03	7.4e-04
Ep: 30, UC: 3000	2.9e-02	1.1e-02	3.3e-03	9.2e-04
Ep: 40, UC: 1000	3.3e-02	1.3e-02	4.4e-03	1.1e-03
Ep: 40, UC: 2000	2.9e-02	1.2e-02	3.9e-03	9.0e-04
Ep: 40, UC: 3000	2.8e-02	1.0e-02	3.4e-03	7.7e-04

Across all settings, the boosted models consistently exhibit lower BER than the baseline models at moderate and high E_b/N_0. The magnitude of improvement varies according to the hyperparameter configuration. In both code families, higher values of `target_uc` generally correspond to lower BER. The effect of increasing `epochs_post` is visible as well, though the improvement increment between 20, 30, and 40 fine-tuning epochs depends on the chosen `target_uc` level.

For the LDPC code (Table I), the largest performance gains appear in the highest `target_uc` setting. The configuration with `epochs_post` $=$ 40 and `target_uc` $=$ 3000 yields the lowest BER across

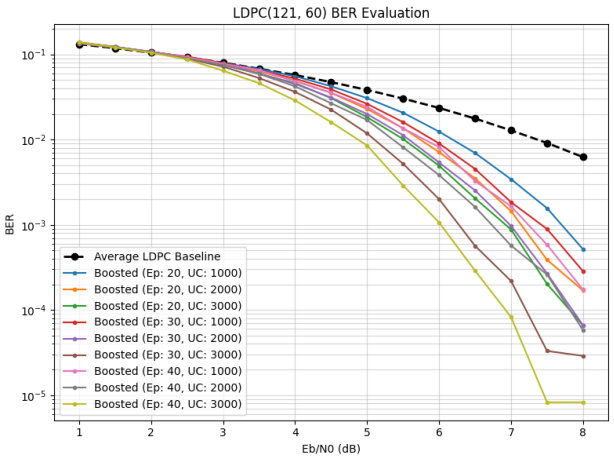

Fig. 1: BER Performance for the LDPC(121, 60) Code.

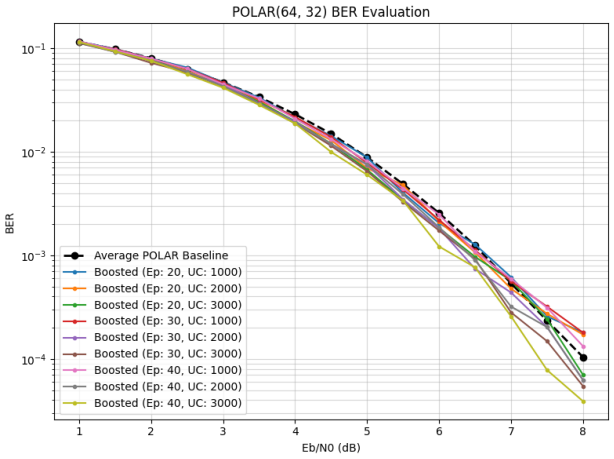

Fig. 2: BER Performance for the Polar(64, 32) Code.

the reported E_b/N_0 samples, including a reduction to 2.9×10^{-4} at $E_b/N_0 = 6.5$. Intermediate configurations show corresponding intermediate BER values, forming a generally monotonic pattern with respect to both hyperparameters.

For the Polar code (Table II), the trends align with those seen in the LDPC experiments, though the performance gain is less pronounced. At each reported E_b/N_0, BER decreases as `target_uc` increases, and mild improvements are also observed as `epochs_post` increases. Among all configurations tested, the lowest BER values occur for `epochs_post` $=$ 40 and `target_uc` $=$ 3000, reaching 7.7×10^{-4} at $E_b/N_0 = 6.5$.

The BER curves in Figures 1 and 2 reinforce these observations. For the LDPC code, the gap between boosted and baseline models widens steadily across increasing E_b/N_0, with the highest `target_uc` con-

figurations forming a clearly separated lower envelope. For the Polar code, the boosted curves show less divergence from the baseline compared to those for the LDPC code, and certain curves (such as the Ep: 30 | UC: 1000 configuration) exhibit a worse performance than the baseline for specific SNRs; nevertheless, the boosted curves still perform better than the baseline for the most part.

Overall, the numerical results show consistent BER reductions for both code types across all hyperparameter configurations evaluated, with `target_uc` providing the largest variation in measurable performance outcomes.

V. CONCLUSION

To recap, this paper examined the empirical impact of several boosting methodologies, originally developed for neural min-sum decoders, when applied to the error-correcting code transformer (ECCT). Four techniques—boosting learning using uncorrected vectors, block-wise training, dynamic weight sharing, and data augmentation—were incorporated into an ECCT fine-tuning pipeline and evaluated on LDPC and polar codes. The experimental framework was implemented through two Google Colab notebooks, enabling reproducible training of baseline models and systematic boosting under controlled hyperparameter sweeps.

Across all tested configurations, boosted ECCT models achieved lower BER than their baseline counterparts at moderate and high values of E_b/N_0. The numerical results indicate that the parameter `target_uc` plays the most significant role in determining the extent of BER reduction for both code families, while increases in `epochs_post` provide additional but comparatively smaller improvements. For the LDPC code, the combined effect of a large uncorrected-vector set and extended fine-tuning yielded the lowest BER values observed in the study. For the polar code, BER reductions were also present, though the degree of improvement was less pronounced.

The collective results demonstrate that the boosting methodologies can be adapted to transformer-based decoders and can provide measurable gains in empirical BER performance. These findings offer initial evidence supporting the claim that the boosting techniques extend beyond neural min-sum decoders. The complete experimental pipeline, including the training and boosting notebooks used to produce the results in this paper, is available in the project repository[3], as well as the full raw evaluation data. Future avenues for research may include broader hyperparameter studies, evaluation on additional code families (such as the Bose–Chaudhuri–Hocquenghem (BCH) code), or the application of boosting methodologies to different code architectures.

[3] `https://github.com/Preenon/Boosting-ECCT`

REFERENCES

[1] Y. Choukroun and L. Wolf, "Error correction code transformer," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, pp. 38695–38705, 2022.

[2] H.-Y. Kwak, D.-Y. Yun, Y. Kim, S.-H. Kim, and J.-S. No, "Boosted neural decoders: Achieving extreme reliability of LDPC codes for 6G networks," *IEEE Journal on Selected Areas in Communications*, vol. 43, no. 4, pp. 1089–1102, Apr. 2025.