FLEXH: CONDITIONAL STRUCTURAL CAPACITY FOR COMPACT TRANSFORMER DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformer-based decoders have achieved strong error-correction performance, but their feed-forward networks (FFNs) remain a major parameter bottleneck. In this paper, we propose FlexH, a plug-in structural preprocessor that addresses this bottleneck by adapting the parity-check representation to each received word. Using channel reliabilities, FlexH prioritizes unreliable positions when selecting linearly independent columns and turns the selected columns into an identity block. This preserves the code and syndrome constraints and provides *conditional structural capacity* without additional trainable parameters. This allows the FFN expansion factor to be reduced from four to two within otherwise unchanged Transformer decoders. Experiments across three Transformer decoders on BCH, Polar, and punctured Reed–Muller (PRM) codes show average reductions of roughly two orders of magnitude in both bit error rate (BER) and block error rate (BLER) at 6 dB. At the same time, parameters and FLOPs decrease by about one third, with lower peak GPU memory. FlexH–EfficientMPT also surpasses Chase-II in nearly all matched BCH comparisons, with BLER approaching available maximum-likelihood (ML) references on BCH and Polar. Ablation studies further support reliability-guided structural adaptation as a practical way to improve decoding performance with smaller FFNs.

1 INTRODUCTION

Recent advances in deep learning have made neural decoders an increasingly effective approach to soft decoding of error-correcting codes. Among them, Transformer-based architectures have attracted particular attention because they can combine flexible feature learning with explicit code constraints. The Error Correction Code Transformer (ECCT) showed that a mask derived from the parity-check matrix (PCM) can expose relationships among codeword bits to self-attention (Choukroun & Wolf, 2022). Building on this line, the Cross-attention Message-Passing Transformer (CrossMPT) and Efficient Message-Passing Transformer (EfficientMPT) made the underlying message passing more explicit and efficient (Park et al., 2025b; 2026). The same lesson appears in recent work on cyclic codes, where code-specific algebraic structure is turned into drop-in priors that improve both decoding performance and parameter efficiency (Xiao et al., 2026). These studies suggest that the representation of code structure is an important part of decoder design, rather than a fixed preprocessing choice.

At the same time, these architectural advances expose a less discussed efficiency imbalance. Attention and message passing have become increasingly lightweight, but the feed-forward network (FFN) in each Transformer block is still commonly inherited from the standard architecture with an expansion factor of four. In a representative six-layer EfficientMPT with an embedding dimension of 128, the FFNs contain 790,272 parameters, accounting for 72.3% of the entire model. After the attention module is streamlined, the FFN becomes the main bottleneck. Yet simply narrowing the FFN can degrade decoding performance. This tension prompts us to reconsider what role the FFN is actually playing in Transformer-based decoding.

Most Transformer-based decoders construct their graph from a code-level parity-check matrix or a fixed set of masks and reuse that representation for every received word. The reliability pattern, however, changes from one received word to another. We hypothesize that the network may need extra capacity to compensate for this mismatch, creating what we call the *fixed-graph capacity burden*.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

This motivates us to consider the parity-check representation and the FFN width jointly, instead of compressing the FFN in isolation.

The Multiple-Masks Error Correction Code Transformer (MM-ECCT) offers a concrete clue for how such an adaptation might be constructed (Park et al., 2025a). With a systematic parity-check matrix, it observes lower bit error rates at positions covered by the identity block than at those associated with the remaining dense block. MM-ECCT therefore uses complementary masks to extend the identity coverage to different groups of positions. This result suggests that the identity block gives its corresponding bits a representation that is easier for the Transformer to handle. However, the identity locations in these masks are still determined at the code level and reused for every received word. Since the reliability pattern varies across received words, it is natural to ask whether the identity block can instead be constructed around the current received word.

Motivated by this observation, we propose FlexH. For each received word, it selects a full-rank column basis by scanning positions from least to most reliable, and uses exact binary row operations to turn the selected columns into an identity block. The same row operations are applied to the syndrome, so the transformed representation preserves both the original code and the syndrome constraints. The transformation is computed only once per received word and introduces no trainable parameters. As a result, the selected unreliable positions are assigned unit-column roles according to the current observation, rather than according to a fixed coordinate order. With this receiver-conditioned representation, we find that an FFN with an expansion factor of two can match or outperform the original factor-four decoder. We refer to this input-dependent flexibility as *conditional structural capacity*. Our focus is on using receiver-conditioned parity-check reparameterization to support Transformer decoding with smaller FFNs.

We evaluate this idea through structure–width and basis-selection ablations, together with experiments across multiple code families and Transformer decoder backbones. The results consistently show that receiver-conditioned reparameterization allows a narrower standard FFN to outperform the wider fixed-representation baseline. They also show that the gain does not come from an arbitrary equivalent parity-check matrix, but from aligning the identity representation with the reliability of the current received word. These findings support the use of adaptive algebraic structure as a practical way to reduce the parametric capacity required by Transformer-based decoders.

Our key contributions are summarized as follows:

- We propose FlexH, a parameter-free receiver-conditioned reparameterization that turns a reliability-guided set of linearly independent columns into an identity block while exactly preserving the code and syndrome constraints.
- We demonstrate that receiver-conditioned parity-check representations support smaller FFNs in Transformer decoding: FlexH with an FFN expansion factor of two outperforms standard fixed-graph decoders with an expansion factor of four. This reframes compact decoding as a joint design of algebraic structure and model capacity, rather than standalone FFN compression.
- With its FFN expansion factor set to two, FlexH reduces BER and BLER by roughly two orders of magnitude on average at 6 dB relative to the corresponding fixed-graph decoders, with gains exceeding four orders on PRM codes. FlexH–EfficientMPT surpasses Chase-II (Chase, 1972) in nearly all matched BCH comparisons, with BLER approaching available ML references at several BCH and Polar operating points. Under the same compact configuration, FlexH–EfficientMPT uses about 36% fewer parameters and about 35% fewer FLOPs, including FlexH preprocessing, with lower peak inference memory.

2 Background

2.1 Binary Linear Codes and Syndrome Decoding

An $[n, k]$ binary linear code $\mathcal{C}$ is a k-dimensional subspace of $\{0,1\}^n$. Let $\boldsymbol{H} \in \{0,1\}^{(n-k)\times n}$ be a full-row-rank parity-check matrix. Then $\mathcal{C} = \{\boldsymbol{x} \in \{0,1\}^n : \boldsymbol{H}\boldsymbol{x} = \boldsymbol{0}\}$, with all binary-vector and matrix operations performed over $\mathrm{GF}(2)$. Under binary phase-shift keying (BPSK), a codeword $\boldsymbol{x}$ is mapped to $\boldsymbol{x}_s = \boldsymbol{1} - 2\boldsymbol{x}$ and transmitted over an additive white Gaussian noise (AWGN) channel. The received vector is $\boldsymbol{y} \in \mathbb{R}^n$, where $\boldsymbol{y} = \boldsymbol{x}_s + \boldsymbol{z}$ and $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2\boldsymbol{I})$. The magnitude $|y_i|$ provides

a direct reliability measure for the i-th hard decision: the closer it is to zero, the less reliable the corresponding position is.

Let $\boldsymbol{y}_b = \mathrm{bin}(\mathrm{sign}(\boldsymbol{y})) \in \{0,1\}^n$ denote the hard-decision vector, where $\mathrm{bin}(\cdot)$ maps -1 to 1 and $+1$ to 0. Its syndrome is $\boldsymbol{s}(\boldsymbol{y}) = \boldsymbol{H}\boldsymbol{y}_b$. When $\boldsymbol{s}(\boldsymbol{y}) = \boldsymbol{0}$, the hard decision is a valid codeword; when $\boldsymbol{s}(\boldsymbol{y}) \neq \boldsymbol{0}$, it records the parity constraints that are currently violated. Syndrome-based neural decoding uses $|\boldsymbol{y}|$ together with $\boldsymbol{s}(\boldsymbol{y})$ instead of asking the network to learn the whole codebook directly. This exploits the symmetry of a linear code and improves generalization across transmitted codewords (Bennatan et al., 2018).

A code does not have a unique parity-check matrix. For any invertible $\boldsymbol{A} \in \{0,1\}^{(n-k)\times(n-k)}$, the row-equivalent matrix $\boldsymbol{H}' = \boldsymbol{A}\boldsymbol{H}$ describes exactly the same code, and its consistent syndrome is $\boldsymbol{s}'(\boldsymbol{y}) = \boldsymbol{A}\boldsymbol{s}(\boldsymbol{y})$. Since $\boldsymbol{A}$ is invertible, $\boldsymbol{H}'\boldsymbol{x} = \boldsymbol{0}$ if and only if $\boldsymbol{H}\boldsymbol{x} = \boldsymbol{0}$. Row operations therefore change how the parity constraints are presented, but not the code itself.

2.2 Transformer-Based Neural Decoders and Their FFNs

Transformer-based neural decoders make the code constraints part of the computation rather than using them only as a training signal. ECCT embeds the reliability magnitudes and syndrome values as tokens, and constructs its attention mask from the parity-check matrix (Choukroun & Wolf, 2022). The nonzero pattern of $\boldsymbol{H}$ naturally defines which variable and check nodes are connected. CrossMPT makes the exchange between these two node types explicit through cross-attention (Park et al., 2025b), while EfficientMPT further simplifies the corresponding message-passing operation (Park et al., 2026). Although the implementations differ, all three use $\boldsymbol{H}$ to determine how reliability and syndrome features are exchanged.

Each decoder layer also contains a position-wise feed-forward network. For a token $\boldsymbol{h} \in \mathbb{R}^d$, it takes the standard form $\mathrm{FFN}(\boldsymbol{h}) = \boldsymbol{W}_2\phi(\boldsymbol{W}_1\boldsymbol{h} + \boldsymbol{\beta}_1) + \boldsymbol{\beta}_2$, where $\boldsymbol{W}_1 \in \mathbb{R}^{md\times d}$ expands the feature dimension by a factor m and $\boldsymbol{W}_2 \in \mathbb{R}^{d\times md}$ projects it back to d. The standard Transformer commonly sets $m = 4$ (Vaswani et al., 2017). While message passing aggregates information over the graph induced by $\boldsymbol{H}$, the FFN transforms the features locally at each node. It contains approximately $2md^2$ parameters per layer and therefore accounts for a large share of the parameters once message passing is made lightweight. A final prediction head then produces one correction logit for each codeword position.

The parity-check matrix therefore has two roles in a Transformer decoder. Algebraically, it defines the constraints used to form the syndrome; computationally, it defines the graph over which features are exchanged. Row-equivalent matrices describe the same code but generally induce different graph representations. Most Transformer decoders choose one such representation at the code level, while multi-mask designs predefine a small collection (Park et al., 2025a). In both cases, the same graph or graph set is reused for every received word.

3 FlexH: Receiver-Conditioned Reparameterization

3.1 Overview: FlexH as a Plug-In Preprocessor

FlexH adapts the decoding graph without changing the code. For each received word $\boldsymbol{y} \in \mathbb{R}^n$, it prioritizes unreliable positions when selecting $n-k$ linearly independent columns and transforms the selected columns into an identity block. Operationally, we keep the reliability magnitudes $|\boldsymbol{y}|$ unchanged and replace the fixed structural pair $(\boldsymbol{H}, \boldsymbol{s})$ by a receiver-conditioned pair $(\boldsymbol{H}_{\boldsymbol{y}}, \boldsymbol{s}_{\boldsymbol{y}})$. This transformation is computed once, before any neural layer, and introduces no trainable parameters. Figure 1 shows the complete interface.

This preprocessor is applicable to Transformer-based decoders in general; here we focus on ECCT, CrossMPT, and EfficientMPT. ECCT constructs its usual attention mask from $\boldsymbol{H}_{\boldsymbol{y}}$ instead of $\boldsymbol{H}$. CrossMPT uses $\boldsymbol{H}_{\boldsymbol{y}}$ and $\boldsymbol{H}_{\boldsymbol{y}}^{\mathsf{T}}$ in the same two directed cross-attention masks that originally use $\boldsymbol{H}$ and $\boldsymbol{H}^{\mathsf{T}}$. EfficientMPT makes the same substitution in its parity-check projection and aggregation operations. The token embedding, message-passing or attention blocks, decoder depth, feature dimension, prediction head, training target, and binary cross-entropy loss are unchanged.

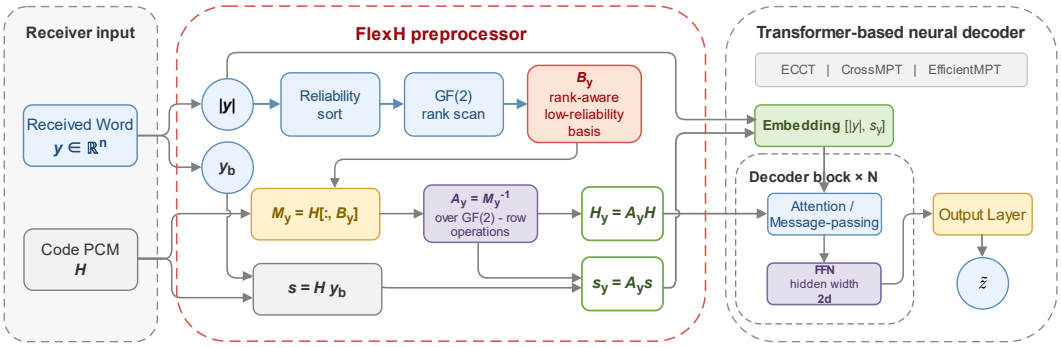


Figure 1: Overview of FlexH. The preprocessor runs once per received word and replaces $(\boldsymbol{H}, \boldsymbol{s})$ with the consistent receiver-conditioned pair $(\boldsymbol{H_y}, \boldsymbol{s_y})$ while keeping $|\boldsymbol{y}|$ unchanged. The neural decoder remains unchanged, except that its FFN width is set to $2d$.

3.2 Receiver-Conditioned Reparameterization and Exactness

Reliability-guided full basis. Let $\boldsymbol{H} \in \{0,1\}^{(n-k)\times n}$ be full row rank, let $\boldsymbol{y}_b \in \{0,1\}^n$ be the hard decision, and let $\boldsymbol{s} = \boldsymbol{H}\boldsymbol{y}_b$. We stably order the coordinates by increasing $|y_i|$, with the coordinate index breaking ties. Starting from an empty set, we scan this order and retain a column only when it increases the rank over $\mathrm{GF}(2)$. The scan stops after $n-k$ columns have been retained, giving the ordered full basis $\mathcal{B}_{\boldsymbol{y}}$. This rank-aware rule is important: simply taking the first $n-k$ coordinates may produce a singular submatrix and cannot define the required row transformation.

Because column independence defines a linear matroid, the greedy scan returns a minimum-sum-magnitude full basis under the stable tie rule. Thus, among all valid full bases, it places the identity representation on coordinates that are collectively as unreliable as possible. The selection uses only $|\boldsymbol{y}|$; neither the hard-decision signs nor the syndrome enter the rule.

Exact receiver-conditioned view. Using the selected submatrix and its binary inverse, we apply the same row transform to the parity-check matrix and syndrome:

$$\boldsymbol{M_y} = \boldsymbol{H}[:, \mathcal{B}_{\boldsymbol{y}}], \qquad \boldsymbol{A_y} = \boldsymbol{M_y}^{-1}, \qquad \boldsymbol{H_y} = \boldsymbol{A_y}\boldsymbol{H}, \qquad \boldsymbol{s_y} = \boldsymbol{A_y}\boldsymbol{s},$$

where all operations are performed over $\mathrm{GF}(2)$.

Figure 2 makes the rank constraint explicit. The three smallest magnitudes belong to $(\boldsymbol{v}_3, \boldsymbol{v}_4, \boldsymbol{v}_6)$, so a reliability-only rule would try them first. However, $\boldsymbol{v}_6 = \boldsymbol{v}_3 + \boldsymbol{v}_4$ over $\mathrm{GF}(2)$; after retaining $\boldsymbol{v}_3$ and $\boldsymbol{v}_4$, adding $\boldsymbol{v}_6$ leaves the rank at two. The resulting submatrix would be singular and could not be inverted. The scan therefore skips $\boldsymbol{v}_6$ and tests the next candidate, $\boldsymbol{v}_5$, which raises the rank to three and gives $\mathcal{B}_{\boldsymbol{y}} = (\boldsymbol{v}_3, \boldsymbol{v}_4, \boldsymbol{v}_5)$.

By construction, $\boldsymbol{H_y}[:, \mathcal{B}_{\boldsymbol{y}}] = \boldsymbol{I}_{n-k}$. Hence, the $n-k$ retained unreliable coordinates become distinct unit columns. No row-equivalent matrix with $n-k$ rows can provide more than $n-k$ linearly independent identity columns, so the construction achieves the maximum possible independent identity coverage. The row transform maps the selected coordinates to the identity without moving any bit coordinate.

FlexH turns the row-equivalent family into a receiver-conditioned representation space: reliability first selects an ordered full basis for the current word, and that basis uniquely determines the transform $\boldsymbol{A_y}$. This differs from using one fixed systematic form, selecting from a code-level collection of pre-enumerated masks, or applying an arbitrary row transform unrelated to the channel observation.

Preservation and preprocessing cost. The adaptation changes the representation, not the decoding problem. Since $\boldsymbol{A_y}$ is invertible, $\mathrm{null}(\boldsymbol{H_y}) = \mathrm{null}(\boldsymbol{H})$, and therefore $\boldsymbol{H_y}$ defines exactly the same code as $\boldsymbol{H}$. The transformed syndrome remains consistent with the transformed graph because $\boldsymbol{s_y} = \boldsymbol{A_y}\boldsymbol{H}\boldsymbol{y}_b = \boldsymbol{H_y}\boldsymbol{y}_b$. More generally, $\boldsymbol{H_y}\boldsymbol{x} = \boldsymbol{s_y}$ if and only if $\boldsymbol{H}\boldsymbol{x} = \boldsymbol{s}$. We use row operations only—never a column permutation—so output coordinate i still corresponds to the original codeword position i.

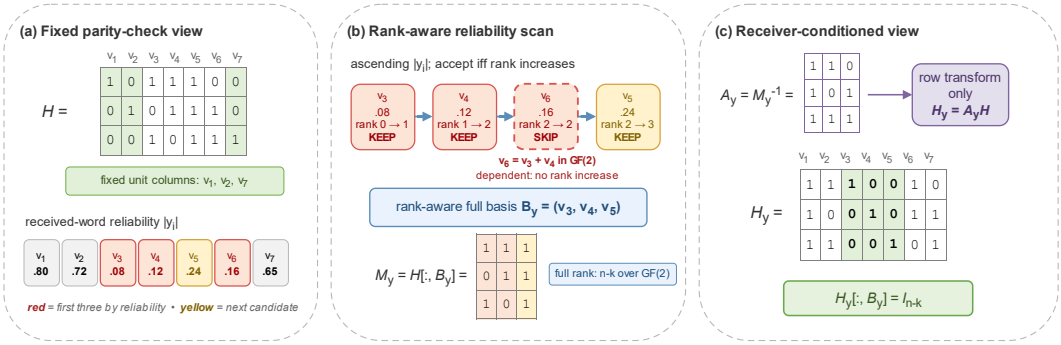


Figure 2: A $(7, 4)$ Hamming-code example with a received vector $\boldsymbol{y}$. Its entries are sorted by reliability; red marks the least reliable positions, and yellow marks the next candidate.

The preprocessor consists of a stable sort and incremental binary elimination. Sorting costs $\mathcal{O}(n \log n)$, while a straightforward rank scan and inverse cost $\mathcal{O}(n(n-k)^2)$ bit operations; the inverse can be accumulated during the same elimination pass. This one-shot work is performed once per received word, not once per decoder layer.

We developed FlexH around representations that match the decoding graph to each received word, and unexpectedly found that it uses matrix operations similar to those of adaptive belief propagation (ABP) (Jiang & Narayanan, 2006). Whereas ABP alternates matrix adaptation with BP-based reliability updates, FlexH constructs a consistent $(\boldsymbol{H_y}, \boldsymbol{s_y})$ from channel observations and keeps it fixed across neural layers. The decoder uses this input-dependent structural interface during both training and evaluation. We examine its interaction with Transformer FFN width through matched structure–width ablations, FFN-update analyses, and cross-architecture experiments.

3.3 From Structural Adaptation to FFN Width

The *fixed-graph capacity burden* describes the potential cost of using the same parity-check graph for words with different unreliable coordinates. Message passing therefore uses a sample-invariant assignment of sparse and dense structural roles, which may require additional processing in the subsequent FFN. A wide FFN may consequently be compensating not only for the intrinsic difficulty of decoding, but also for an input graph that is poorly aligned with the current reliability pattern. Reducing FFN width alone saves parameters without addressing that mismatch.

FlexH changes where sample dependence is introduced. The row-equivalent family of $\boldsymbol{H}$ provides many exact computational graphs for the same code, and $|\boldsymbol{y}|$ constructs one member for the current word. In that view, the least-reliable independent coordinates receive distinct unit-column roles before any learned feature transformation. When unreliable positions lie in more complex local graph structures, the features produced by message aggregation may require additional processing by the FFN. Assigning these positions simpler local connections may reduce this processing burden. We interpret this input-dependent family of code-equivalent graphs as *conditional structural capacity* available to the shared decoder without additional trainable parameters. Pairing FlexH with a $2d$ FFN is the compact realization of this view, rather than an isolated width-reduction step applied to the original fixed graph.

This interpretation gives two directly testable predictions for compact decoding. First, a FlexH decoder with a $2d$ FFN can surpass the standard Static-H decoder with a $4d$ FFN. Second, FlexH with a $2d$ FFN should retain performance close to its $4d$ counterpart. The ablation study in Section 4.4 tests these predictions, examines how the FFN updates internal features, and uses basis-selection variants to separate receiver conditioning from generic graph variation.

4 Experiments

We evaluate nine BCH, Polar, and PRM codes with ECCT, CrossMPT, and EfficientMPT as base decoders.

4.1 Experimental Setup and Baselines

The baselines are their fixed-graph versions, Hyper-BP (Nachmani & Wolf, 2019), unrolled for five BP iterations during training and run for 50 shared-weight iterations at inference, Chase-II (Chase, 1972) as a strong specialized decoder for BCH codes, recursive projection–aggregation (RPA) (Ye & Abbe, 2020) as a strong specialized decoder for PRM codes, and ABP (Jiang & Narayanan, 2006), a strong BP-based performance baseline, with at most 20 iterations and $\alpha = 0.1$. We also include the circulant-prior CrossMPT of Xiao et al. (2026), denoted Cyclic CrossMPT, as a recent strong neural baseline tailored to BCH and PRM cyclic codes. We use its $d = 2n$ results. We also report the available maximum-likelihood (ML) references for BCH and Polar from the channel-code database (Helmling et al., 2025).

For our matched base/FlexH comparisons, all Transformer-based decoders start from the same systematic parity-check matrix for each code. We use this strong baseline representation because prior work has shown that systematic masking can improve Transformer-based decoding performance (Park et al., 2025a). The fixed-graph baselines retain this matrix, while FlexH adapts its identity-column locations to each received word. Appendix A gives the exactness proofs, and Appendix B details the preprocessing algorithm and its integration with each decoder.

Following standard experimental settings for Transformer-based decoders, the models in these paired comparisons use $N = 6$ layers, a feature dimension of $d = 128$, and $h = 8$ attention heads. They are trained from scratch for 1000 epochs, with 1000 minibatches per epoch and 128 samples per minibatch. We use Adam with an initial learning rate of 10^{-4} and cosine annealing to 10^{-6}. Training uses all-zero codewords over $E_b/N_0 \in \{2, 3, 4, 5, 6, 7\}$ dB and ordinary binary cross-entropy. We report test results from 1 to 6 dB. At each SNR, sampling continues beyond 10^5 received words until more than 100 block errors are observed, with a maximum of 10^9 words. Training and evaluation are conducted on an NVIDIA GeForce RTX 5090 GPU, and peak inference memory is profiled at batch size 2048. Additional training, evaluation, and profiling details are given in Appendix H.

Table 1: BER and BLER comparison across nine codes at 4–6 dB. Entries are negative natural logarithms of the named error rate. Within each base/FlexH pair, the larger value is bold.

Decoder		ECCT			FlexH–ECCT			CrossMPT			FlexH–CrossMPT			EfficientMPT			FlexH–EfficientMPT		
Code	**Metric**	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6
BCH(63,36)	$-\ln$(BER)	5.18	7.01	9.61	**8.94**	**12.49**	**16.39**	5.09	6.88	9.58	**9.26**	**13.08**	**16.82**	4.92	6.63	9.29	**9.49**	**13.35**	**17.49**
	$-\ln$(BLER)	2.59	4.22	6.62	**6.51**	**9.63**	**12.91**	2.56	4.21	6.78	**7.04**	**10.41**	**13.24**	2.35	3.91	6.37	**7.34**	**11.01**	**14.00**
BCH(63,45)	$-\ln$(BER)	5.56	7.71	10.65	**8.09**	**11.66**	**16.07**	5.88	8.18	11.49	**8.14**	**11.69**	**15.76**	5.72	7.98	11.24	**8.15**	**11.78**	**16.17**
	$-\ln$(BLER)	2.70	4.69	7.49	**5.69**	**9.08**	**13.20**	3.15	5.34	8.63	**5.77**	**9.19**	**12.65**	2.94	5.08	8.24	**5.80**	**9.42**	**13.55**
BCH(63,51)	$-\ln$(BER)	5.65	7.83	10.94	**6.74**	**9.45**	**13.27**	5.78	8.05	11.32	**6.74**	**9.48**	**13.25**	5.68	7.93	11.18	**6.74**	**9.45**	**13.24**
	$-\ln$(BLER)	2.74	4.82	7.73	**4.07**	**6.71**	**10.65**	2.95	5.14	8.29	**4.07**	**6.77**	**10.52**	2.79	4.93	8.01	**4.06**	**6.74**	**10.60**
BCH(127,99)	$-\ln$(BER)	4.56	6.43	9.32	**8.35**	**12.11**	**16.84**	4.59	6.50	9.41	**8.81**	**13.28**	**18.06**	4.50	6.31	9.11	**8.56**	**12.76**	**17.37**
	$-\ln$(BLER)	1.23	2.91	5.64	**5.21**	**8.69**	**12.93**	1.30	3.04	5.81	**5.84**	**10.05**	**13.97**	1.19	2.82	5.47	**5.55**	**9.46**	**13.58**
Polar(64,48)	$-\ln$(BER)	6.38	8.53	10.94	**6.65**	**8.75**	**11.18**	6.51	8.69	11.13	**6.65**	**8.76**	**11.19**	6.39	8.52	11.05	**6.66**	**8.75**	**11.20**
	$-\ln$(BLER)	3.27	5.30	7.74	**3.78**	**5.88**	**8.32**	3.56	5.66	8.15	**3.76**	**5.85**	**8.33**	3.37	5.41	7.95	**3.78**	**5.89**	**8.31**
Polar(128,96)	$-\ln$(BER)	6.36	9.19	12.53	**8.01**	**10.70**	**13.33**	6.58	9.52	12.80	**8.08**	**10.71**	**13.28**	6.26	9.07	12.41	**8.07**	**10.75**	**13.30**
	$-\ln$(BLER)	2.60	5.17	8.39	**4.65**	**7.21**	**9.72**	2.95	5.63	8.69	**4.76**	**7.22**	**9.72**	2.62	5.16	8.29	**4.72**	**7.20**	**9.59**
PRM(63,42)	$-\ln$(BER)	6.29	8.88	12.34	**8.27**	**11.48**	**15.43**	6.32	8.81	12.39	**8.33**	**11.63**	**15.46**	5.87	8.17	11.38	**8.34**	**11.55**	**15.54**
	$-\ln$(BLER)	3.42	5.78	9.09	**5.91**	**9.04**	**12.75**	3.67	6.05	9.47	**5.96**	**9.24**	**12.28**	3.13	5.28	8.31	**6.02**	**9.24**	**12.99**
PRM(127,64)	$-\ln$(BER)	3.73	5.11	7.22	**9.68**	**13.08**	**16.77**	3.75	5.18	7.38	**10.23**	**14.07**	**17.63**	3.45	4.62	6.49	**9.74**	**13.15**	**16.76**
	$-\ln$(BLER)	0.84	1.89	3.66	**6.81**	**9.90**	**12.50**	0.96	2.12	4.10	**7.71**	**11.09**	**13.25**	0.60	1.49	3.08	**7.08**	**10.16**	**12.97**
PRM(127,99)	$-\ln$(BER)	4.96	7.21	10.46	**8.17**	**11.96**	**16.73**	5.18	7.60	11.22	**8.35**	**12.22**	**17.04**	5.01	7.26	10.74	**8.26**	**12.15**	**17.06**
	$-\ln$(BLER)	1.50	3.48	6.44	**5.09**	**8.70**	**13.07**	1.87	4.07	7.52	**5.34**	**9.13**	**13.57**	1.63	3.66	7.00	**5.27**	**9.03**	**13.56**

4.2 From Bit Accuracy to Reliable Codewords

BER measures individual bit errors, whereas BLER measures failures to recover an entire codeword. For applications requiring complete-codeword recovery, BLER is particularly important. FlexH improves both metrics, with its BLER gains directly reflecting more reliable codeword recovery.

Across the full 1–6 dB range, FlexH improves both BER and BLER for every matched decoder–code pair. At 6 dB, the average reduction in each metric is roughly two orders of magnitude relative to the corresponding fixed-graph base decoder. FlexH–EfficientMPT outperforms Hyper-BP in both metrics throughout the tested range. It also surpasses ABP and Chase-II in nearly all matched comparisons, and achieves performance comparable to RPA on PRM(63,42).

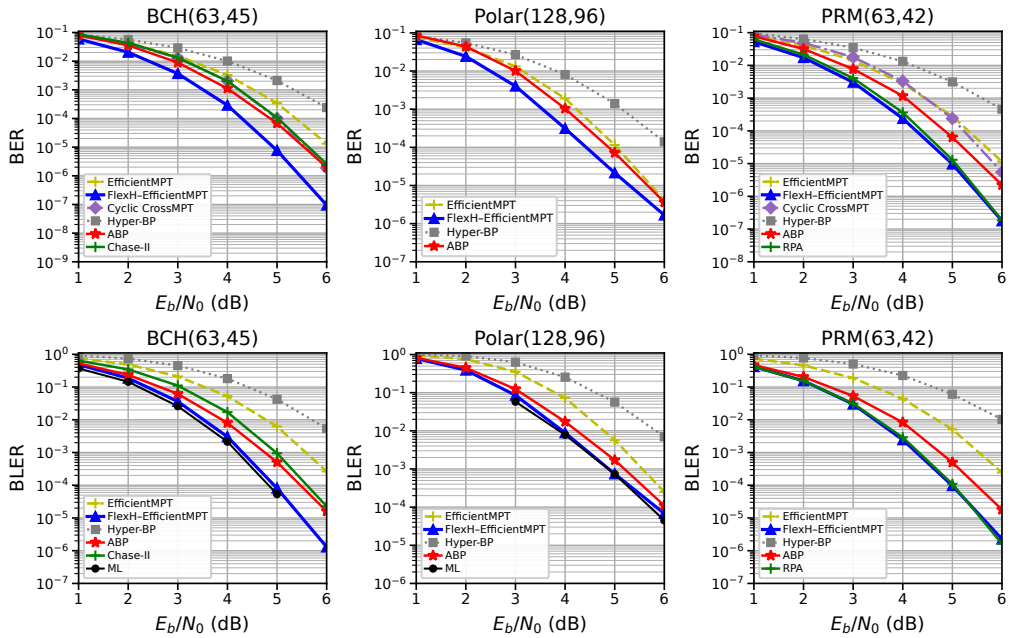


Figure 3: BER (top) and BLER (bottom) on representative BCH, Polar, and PRM codes from 1 to 6 dB. EfficientMPT and FlexH–EfficientMPT are compared with Cyclic CrossMPT (BER), Hyper-BP, ABP, Chase-II for BCH, and RPA for PRM.

FlexH–EfficientMPT approaches the available ML BLER references on BCH and Polar: on BCH(63,51) at 4 dB, its BLER is 1.73×10^{-2} versus the ML value 1.43×10^{-2}, while on Polar(128,96) at 5 dB the two values are 7.43×10^{-4} and 7.36×10^{-4}. The largest gain occurs on PRM(127,64), where FlexH–EfficientMPT reduces BER from 1.52×10^{-3} to 5.26×10^{-8} and BLER from 4.58×10^{-2} to 2.32×10^{-6}, improving both metrics by more than four orders of magnitude.

FlexH–CrossMPT also improves BER over Cyclic CrossMPT at every shared SNR from 1 to 6 dB. Appendix C provides the full EfficientMPT BER/BLER curves and baseline-comparison tables for all three neural decoders, including the available ML references. Further BCH comparisons and long-block BCH/Polar results are collected in Appendix D.

4.3 Accuracy–Efficiency Trade-offs

Across the nine codes, FlexH–ECCT and FlexH–CrossMPT use about 67%–68% of their respective baseline parameters, while FlexH–EfficientMPT uses about 64%. Relative to ECCT, CrossMPT, and EfficientMPT, respectively, the FlexH variants require about 70%–74%, 68%–70%, and 64%–66% of the FLOPs, including preprocessing. Peak inference memory falls to about 32%–39%, 43%–51%, and 69%–88% of the corresponding baselines. Together, these results show that FlexH improves word-level reliability by up to several orders while reducing parameters, FLOPs, and GPU memory. Figure 4 shows this joint improvement on six representative codes: FlexH–EfficientMPT achieves lower BLER with fewer parameters and fewer FLOPs in every pair. Appendix E provides the corresponding memory comparison and extends the efficiency results to additional BCH codes.

4.4 Cross-Decoder Generalization and Ablation Studies

The gains across three Transformer decoders motivate a closer look at their shared structural interface. Using EfficientMPT, we test the width predictions in Section 3.3, examine internal FFN updates, and compare basis-selection rules. Appendix G reports exploratory LDPC results.

Table 2: Efficiency across the nine codes in Table 1. P: parameters (M); F: FLOPs per codeword (M), measured at batch size one and including preprocessing for FlexH; M: peak allocated GPU memory (GiB) at inference batch size 2048. Percentages are relative to the corresponding baseline (100%). The lower value in each base/FlexH pair is bold.

Code	ECCT			FlexH–ECCT			CrossMPT			FlexH–CrossMPT			EfficientMPT			FlexH–EfficientMPT		
	P	F	M	P	F	M	P	F	M	P	F	M	P	F	M	P	F	M
BCH(63,36)	1.208 (100%)	237.4 (100%)	4.68 (100%)	**0.813** (67%)	**166.8** (70%)	**1.74** (37%)	1.208 (100%)	223.0 (100%)	1.44 (100%)	**0.813** (67%)	**152.2** (68%)	**0.73** (51%)	1.093 (100%)	200.5 (100%)	0.83 (100%)	**0.698** (64%)	**129.7** (65%)	**0.59** (72%)
BCH(63,45)	1.206 (100%)	211.4 (100%)	3.87 (100%)	**0.811** (67%)	**147.9** (70%)	**1.49** (38%)	1.206 (100%)	198.2 (100%)	1.22 (100%)	**0.811** (67%)	**134.5** (68%)	**0.62** (51%)	1.093 (100%)	179.1 (100%)	0.81 (100%)	**0.698** (64%)	**115.4** (64%)	**0.57** (70%)
BCH(63,51)	1.205 (100%)	194.4 (100%)	3.37 (100%)	**0.810** (67%)	**135.5** (70%)	**1.32** (39%)	1.205 (100%)	181.7 (100%)	1.07 (100%)	**0.810** (67%)	**122.8** (68%)	**0.55** (51%)	1.093 (100%)	164.9 (100%)	0.80 (100%)	**0.698** (64%)	**105.9** (64%)	**0.55** (69%)
BCH(127,99)	1.230 (100%)	439.8 (100%)	12.98 (100%)	**0.835** (68%)	**318.8** (72%)	**4.25** (33%)	1.230 (100%)	387.9 (100%)	2.87 (100%)	**0.835** (68%)	**266.0** (69%)	**1.36** (47%)	1.093 (100%)	347.3 (100%)	1.58 (100%)	**0.698** (64%)	**225.4** (65%)	**1.11** (70%)
Polar(64,48)	1.206 (100%)	208.6 (100%)	3.77 (100%)	**0.811** (67%)	**145.8** (70%)	**1.45** (38%)	1.206 (100%)	195.2 (100%)	1.18 (100%)	**0.811** (67%)	**132.3** (68%)	**0.60** (51%)	1.093 (100%)	176.6 (100%)	0.81 (100%)	**0.698** (64%)	**113.7** (64%)	**0.57** (70%)
Polar(128,96)	1.231 (100%)	456.5 (100%)	13.78 (100%)	**0.837** (68%)	**331.7** (73%)	**4.48** (33%)	1.231 (100%)	403.0 (100%)	3.09 (100%)	**0.837** (68%)	**277.2** (69%)	**1.45** (47%)	1.093 (100%)	360.0 (100%)	1.59 (100%)	**0.698** (64%)	**234.1** (65%)	**1.13** (71%)
PRM(63,42)	1.206 (100%)	220.0 (100%)	4.13 (100%)	**0.812** (67%)	**154.1** (70%)	**1.57** (38%)	1.206 (100%)	206.5 (100%)	1.29 (100%)	**0.812** (67%)	**140.4** (68%)	**0.66** (51%)	1.093 (100%)	186.2 (100%)	0.81 (100%)	**0.698** (64%)	**120.2** (65%)	**0.58** (71%)
PRM(127,64)	1.239 (100%)	559.6 (100%)	19.14 (100%)	**0.844** (68%)	**412.2** (74%)	**6.05** (32%)	1.239 (100%)	497.8 (100%)	4.87 (100%)	**0.844** (68%)	**348.4** (70%)	**2.10** (43%)	1.093 (100%)	437.9 (100%)	1.65 (100%)	**0.698** (64%)	**288.5** (66%)	**1.46** (88%)
PRM(127,99)	1.230 (100%)	439.8 (100%)	12.98 (100%)	**0.835** (68%)	**318.8** (72%)	**4.25** (33%)	1.230 (100%)	387.9 (100%)	2.87 (100%)	**0.835** (68%)	**266.0** (69%)	**1.36** (47%)	1.093 (100%)	347.3 (100%)	1.58 (100%)	**0.698** (64%)	**225.4** (65%)	**1.11** (70%)

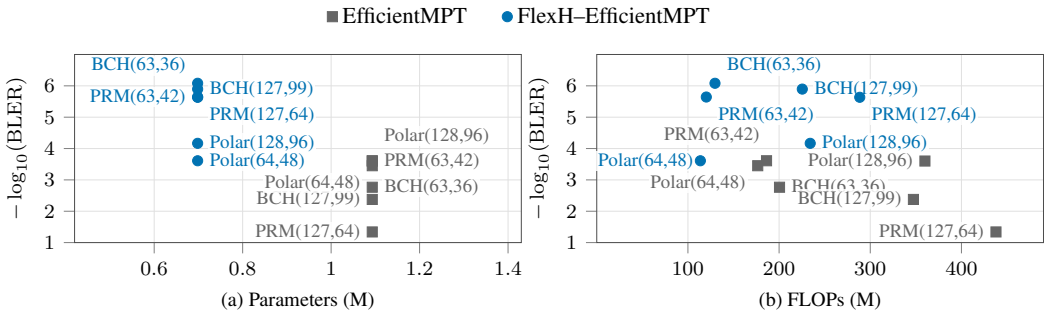


Figure 4: BLER versus (a) parameters and (b) FLOPs at 6 dB on six BCH, Polar, and PRM codes, comparing EfficientMPT with FlexH–EfficientMPT. FlexH FLOPs include preprocessing.

Table 3: Ablation studies with EfficientMPT at $E_b/N_0 = 6$ dB. Entries report $-\ln(\text{BER})/-\ln(\text{BLER})$. The upper block tests structure and FFN width; the lower block tests column priorities. Static-H keeps the systematic PCM fixed. Random-full uses a fresh random order for each word; Most-reliable-full scans from most to least reliable; Magnitude + syndrome favors bits with a larger fraction of unsatisfied incident checks at equal normalized magnitude; FlexH scans from least to most reliable. The three basis controls keep FlexH's $2d$ architecture, training setup, rank test, and exact row transformation. Within each block, the best result for each code and metric is bold.

Control	Representation	FFN	BCH(127,99)	Polar(128,96)	PRM(127,64)
Structure–width	Static-H	$4d$	9.11 / 5.47	12.41 / 8.29	6.49 / 3.08
	Static-H	$2d$	8.48 / 4.71	11.66 / 7.84	5.98 / 2.96
	FlexH	$2d$	17.37 / 13.58	**13.30** / 9.59	16.76 / **12.97**
	FlexH	$4d$	**17.77** / **13.66**	13.24 / **9.69**	**17.01** / 12.93
Basis selection	Static-H	$4d$	9.11 / 5.47	12.41 / 8.29	6.49 / 3.08
	Random-full	$2d$	9.11 / 5.46	10.06 / 6.13	7.28 / 3.83
	Most-reliable-full	$2d$	8.12 / 4.45	8.99 / 5.04	13.62 / 8.85
	Magnitude + syndrome	$2d$	16.88 / 12.95	**13.39** / **9.73**	16.69 / 12.87
	FlexH	$2d$	**17.37** / **13.58**	13.30 / 9.59	**16.76** / **12.97**

Structure and FFN capacity. At 6 dB, narrowing Static-H from $4d$ to $2d$ worsens BER and BLER on all three codes, whereas FlexH $2d$ outperforms both Static-H widths. Its BLER is roughly three

and a half orders lower than Static-H $4d$ on BCH, half an order on Polar, and over four orders on PRM. FlexH $2d$ and $4d$ perform similarly, supporting both predictions in Section 3.3.

Figure 5 examines the FFN updates added to bit features in the second EfficientMPT block, averaged over 4–6 dB on these codes. Update size is measured by vector length.

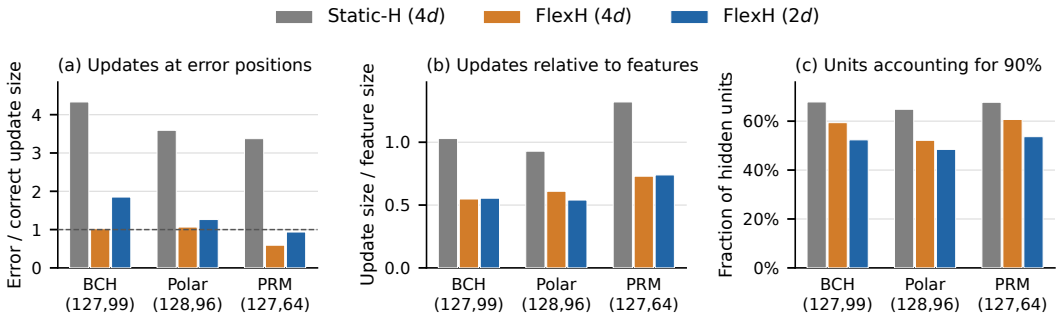


Figure 5: FFN updates to bit features in the second EfficientMPT block, averaged over 4–6 dB. Update size is measured by vector length. (a) Mean update size at initially incorrect positions relative to correct positions; the dashed line indicates equal means. (b) Update size relative to the feature size before addition. (c) Hidden-unit fraction accounting for 90% of estimated output contributions.

In Figure 5(a), the ratio drops from 3.4–4.3 for Static-H $4d$ to 0.6–1.1 for FlexH $4d$: the FFN no longer assigns much larger updates to initially incorrect positions. Narrowing FlexH to $2d$ raises the ratio, but it remains below Static-H's. In Figure 5(b), the relative update size falls by 34%–47% at matched $4d$ width; on BCH, it decreases from about 1.03 to about 0.55 for both FlexH widths. This means that the FFN makes smaller adjustments relative to the existing features in this block, and narrowing it does not reverse this pattern. In Figure 5(c), the hidden-unit fraction falls from 65%–68% for Static-H $4d$ to 52%–61% for FlexH $4d$ and 48%–54% for FlexH $2d$. Thus, a smaller share of units carries most of the estimated output contribution, and this concentration persists after narrowing. These changes already appear at matched $4d$ width, so they cannot be explained by FFN narrowing alone.

Together with the performance in Table 3, these observations are consistent with a reduced fixed-graph capacity burden, supporting our conditional structural capacity interpretation. The update statistics alone do not establish how much width can be removed. Appendix F.1 gives the metric definitions and all-layer results.

Reliability-conditioned basis selection. At 6 dB, Random-full is close to Static-H on BCH, improves PRM modestly, and degrades Polar. Most-reliable-full also degrades BCH and Polar; even on PRM, its BLER remains nearly two orders higher than FlexH's. Both controls retain FlexH's $2d$ FFN and the same identity-block size. Thus, neither graph variation alone nor prioritizing reliable positions reproduces FlexH's benefit. Magnitude + syndrome improves Polar slightly but worsens BCH and PRM, offering no consistent advantage over the simpler magnitude-only rule. Appendix F provides the scoring details and additional BER ablation curves.

5 Conclusion

We presented FlexH, a parameter-free preprocessor that adapts parity-check representations to received words while preserving code and syndrome constraints. Experiments across three Transformer decoders show consistent improvements in BER and, crucially, BLER, with reductions reaching several orders of magnitude on BCH and PRM. FlexH–EfficientMPT outperforms ABP and surpasses Chase-II in nearly all matched BCH comparisons, while approaching the available ML references on BCH and Polar. These gains are achieved with narrower FFNs, fewer parameters, lower FLOPs, and lower peak GPU memory. Ablation studies support conditional structural capacity as a practical basis for compact decoding. Overall, adapting code-structure representations to the received word can improve both the performance and efficiency of neural decoders.

AI USE STATEMENT

Generative AI tools were used to assist with manuscript organization, language drafting and editing, literature-search planning, and code review. All AI-assisted text, mathematical claims, citations, experimental interpretations, and code changes are reviewed and verified by the authors. The authors take responsibility for the final content of this work, including all text, claims, and artifacts produced with AI assistance.

ETHICS STATEMENT

This work studies algorithms for error-correcting-code decoding using simulated communication channels and does not involve human subjects or personal data. We do not identify specific societal risks beyond those associated with the deployment context of communication systems.

REPRODUCIBILITY STATEMENT

Section 3 specifies the proposed transformation, and Appendix A proves its algebraic properties. Appendix B documents the construction and its integration with the base decoders. Section 4.1 and Appendix H document the model, training, evaluation, and profiling settings. Additional results, complete tables, efficiency analyses, ablations, and long-block and sparse-code extensions are provided in Appendices C–G.

REFERENCES

Amir Bennatan, Yoni Choukroun, and Pavel Kisilev. Deep learning for decoding of linear codes - a syndrome-based approach. *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 1595–1599, 2018.

Andreas Buchberger, Christian Häger, Henry D. Pfister, Laurent Schmalen, and Alexandre Graell i Amat. Learned decimation for neural belief propagation decoders : Invited paper. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8273–8277, 2021. doi: 10.1109/ICASSP39728.2021.9414407.

Avi Caciularu, Nir Raviv, Tomer Raviv, Jacob Goldberger, and Yair Be'ery. perm2vec: Attentive graph permutation selection for decoding of error correction codes. *IEEE Journal on Selected Areas in Communications*, 39(1):79–88, 2021. doi: 10.1109/JSAC.2020.3036951.

Sebastian Cammerer, Tobias Gruber, Jakob Hoydis, and Stephan ten Brink. Scaling deep learning-based decoding of polar codes via partitioning. In *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pp. 1–6, 2017. doi: 10.1109/GLOCOM.2017.8254811.

D. Chase. Class of algorithms for decoding block codes with channel measurement information. *IEEE Transactions on Information Theory*, 18(1):170–182, 1972. doi: 10.1109/TIT.1972.1054746.

Xiangyu Chen and Min Ye. Cyclically equivariant neural decoders for cyclic codes. *CoRR*, abs/2105.05540, 2021. URL https://arxiv.org/abs/2105.05540.

Yoni Choukroun and Lior Wolf. Error correction code transformer. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 38695–38705. Curran Associates, Inc., 2022.

Yoni Choukroun and Lior Wolf. Denoising diffusion error correction codes. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=rLwC0_MG-4w.

Yoni Choukroun and Lior Wolf. A foundation model for error correction codes. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=7KDuQPrAF3.

Yoni Choukroun and Lior Wolf. Learning linear block error correction codes, 2024b. URL https://arxiv.org/abs/2405.04050.

Tobias Gruber, Sebastian Cammerer, Jakob Hoydis, and Stephan ten Brink. On deep learning-based channel decoding. In *2017 51st Annual Conference on Information Sciences and Systems (CISS)*, pp. 1–6, 2017. doi: 10.1109/CISS.2017.7926071.

Michael Helmling, Stefan Scholl, Florian Gensheimer, Tobias Dietz, Kira Kraft, Oliver Griebel, Stefan Ruzika, and Norbert Wehn. Database of Channel Codes and ML Simulation Results. `www.rptu.de/channel-codes`, 2025.

Mohamed Ibnkahla. Applications of neural networks to digital communications - a survey. *Signal Process.*, 80:1185–1215, 2000.

Mohammad Vahid Jamali, Hamid Saber, Homayoon Hatami, and Jung Hyun Bae. Productae: Toward training larger channel codes based on neural product codes. In *ICC 2022 - IEEE International Conference on Communications*, pp. 3898–3903, 2022. doi: 10.1109/ICC45855.2022.9839215.

J. Jiang and K.R. Narayanan. Iterative soft-input soft-output decoding of reed–solomon codes by adapting the parity-check matrix. *IEEE Transactions on Information Theory*, 52(8):3746–3756, 2006. doi: 10.1109/TIT.2006.878176.

E. Uras Kargı and Tolga M. Duman. A deep learning based decoder for concatenated coding over deletion channels. In *ICC 2024 - IEEE International Conference on Communications*, pp. 2797–2802, 2024. doi: 10.1109/ICC51166.2024.10622561.

Hyeji Kim, Yihan Jiang, Ranvir Rana, Sreeram Kannan, Sewoong Oh, and Pramod Viswanath. Communication algorithms via deep learning. *ArXiv*, abs/1805.09317, 2018.

Cheuk Ting Leung, Mehul Motani, and Rajshekhar V Bhat. Multi-label and concatenated neural block decoders. In *2020 IEEE International Symposium on Information Theory (ISIT)*, pp. 274–279, 2020. doi: 10.1109/ISIT44484.2020.9174106.

Jinyuan Li, Shuai Xiao, and Weijun Fang. ProductMPT: Message-passing transformer for product codes. *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, 2026. to appear.

Loren Lugosch and Warren J. Gross. Neural offset min-sum decoding. In *2017 IEEE International Symposium on Information Theory (ISIT)*, pp. 1361–1365, 2017. doi: 10.1109/ISIT.2017.8006751.

Eliya Nachmani and Lior Wolf. Hyper-graph-network decoders for block codes. *ArXiv*, abs/1909.09036, 2019.

Eliya Nachmani, Yair Be'ery, and David Burshtein. Learning to decode linear codes using deep learning. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 341–346, 2016. doi: 10.1109/ALLERTON.2016.7852251.

Eliya Nachmani, Elad Marciano, Loren Lugosch, Warren J. Gross, David Burshtein, and Yair Be'ery. Deep learning methods for improved decoding of linear codes. *IEEE Journal of Selected Topics in Signal Processing*, 12(1):119–131, 2018. doi: 10.1109/JSTSP.2017.2788405.

Seong-Joon Park, Hee-Youl Kwak, Sang-Hyo Kim, Sunghwan Kim, Yongjune Kim, and Jong-Seon No. Multiple-masks error correction code transformer for short block codes. *IEEE Journal on Selected Areas in Communications*, pp. 1–1, 2025a. doi: 10.1109/JSAC.2025.3559154.

Seong-Joon Park, Hee-Youl Kwak, Sang-Hyo Kim, Yongjune Kim, and Jong-Seon No. Crossmpt: Cross-attention message-passing transformer for error correcting codes. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 56710–56728, 2025b. URL `https://proceedings.iclr.cc/paper_files/paper/2025/file/8f0942c43fcfba4cc66a859b9fcb1bba-Paper-Conference.pdf`.

Seong-Joon Park, Taewoo Park, Hee-Youl Kwak, Sang-Hyo Kim, Yongjune Kim, and Jong-Seon No. Efficient message-passing transformer for error correcting codes. In *The Fourteenth International Conference on Learning Representations*, 2026. URL `https://openreview.net/forum?id=Xk8cwnwu2e`.

Julian Streit, Franziska Weindel, and Reinhard Heckel. Transformer-based decoding in concatenated coding schemes under synchronization errors. In *2025 IEEE International Symposium on Information Theory (ISIT)*, pp. 1–6, 2025. doi: 10.1109/ISIT63088.2025.11195598.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Shuai Xiao, Weijun Fang, and Bin Chen. Transformer-based decoders for cyclic codes: A tanner cycle-equivalent approach. *IEEE Transactions on Communications*, 73(12):13173–13185, 2025. doi: 10.1109/TCOMM.2025.3621239.

Shuai Xiao, Weijun Fang, and Qiaosheng Zhang. Drop-in circulant structural priors for transformer decoding of cyclic codes. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=wDFrxk7306.

Min Ye and Emmanuel Abbe. Recursive projection-aggregation decoding of reed-muller codes. *IEEE Transactions on Information Theory*, 66(8):4948–4965, 2020. doi: 10.1109/TIT.2020.2977917.

Qinshan Zhang, Bin Chen, Tianqu Zhuang, Yong Jiang, and Shu-Tao Xia. Section-wise revolving nbp-like decoders for qc-ldpc codes. In *2024 IEEE International Symposium on Information Theory (ISIT)*, pp. 1397–1402, 2024. doi: 10.1109/ISIT57864.2024.10619193.

A Exactness Proofs and Algebraic Properties

This appendix separates the exact algebraic statements from the empirical capacity interpretation in the main text. Let $\boldsymbol{H} \in \{0,1\}^{(n-k)\times n}$ have row rank $n-k$, and perform all operations below over $\mathrm{GF}(2)$.

A.1 Existence and Optimality of the Selected Basis

Proposition 1. *For every received word $\boldsymbol{y} \in \mathbb{R}^n$, the rank-aware scan in Section 3.2 terminates after selecting $n-k$ columns. The selected set $\mathcal{B}_{\boldsymbol{y}}$ is a minimum-weight basis of the column vector matroid of $\boldsymbol{H}$ when column i has weight $|y_i|$.*

Proof. Because $\mathrm{rank}(\boldsymbol{H}) = n-k$, its columns contain at least one independent set of size $n-k$. Whenever the current set has size smaller than $n-k$, some remaining column must increase its rank; otherwise the span of all columns would have dimension smaller than $n-k$. The scan therefore reaches size $n-k$ and the selected submatrix $\boldsymbol{M}_{\boldsymbol{y}} = \boldsymbol{H}[:, \mathcal{B}_{\boldsymbol{y}}]$ is nonsingular.

The independent column sets of a matrix form a linear matroid. The standard matroid greedy theorem states that scanning elements in nondecreasing weight and retaining exactly those that preserve independence produces a minimum-weight basis. Here the weights are $|y_i|$, and the fixed coordinate index resolves ties. Thus $\mathcal{B}_{\boldsymbol{y}}$ minimizes $\sum_{i\in\mathcal{B}} |y_i|$ over every full column basis $\mathcal{B}$ of $\boldsymbol{H}$. □

The result is stronger than simply selecting the $n-k$ smallest magnitudes. Those columns need not be independent, as illustrated by $\boldsymbol{v}_6$ in Figure 2. The rank test is exactly what turns a reliability order into a valid algebraic basis.

A.2 Identity Coverage and Code Preservation

Proposition 2. *Let $\boldsymbol{A}_{\boldsymbol{y}} = \boldsymbol{M}_{\boldsymbol{y}}^{-1}$ and $\boldsymbol{H}_{\boldsymbol{y}} = \boldsymbol{A}_{\boldsymbol{y}}\boldsymbol{H}$. Then $\boldsymbol{H}_{\boldsymbol{y}}[:, \mathcal{B}_{\boldsymbol{y}}] = \boldsymbol{I}_{n-k}$. Moreover, $n-k$ is the largest possible number of linearly independent identity columns in any row-equivalent $(n-k)\times n$ parity-check matrix.*

Proof. Direct substitution gives $\boldsymbol{H}_{\boldsymbol{y}}[:, \mathcal{B}_{\boldsymbol{y}}] = \boldsymbol{M}_{\boldsymbol{y}}^{-1}\boldsymbol{M}_{\boldsymbol{y}} = \boldsymbol{I}_{n-k}$. Any $(n-k)\times n$ matrix has column rank at most $n-k$, so it cannot contain more than $n-k$ linearly independent unit columns. The construction therefore reaches the maximum possible independent identity coverage. □

This statement concerns independent identity roles, not the total number of columns that happen to repeat a unit vector. Repeated unit columns do not provide additional independent roles.

Proposition 3. *For every* $\boldsymbol{x} \in \{0,1\}^n$,

$$\boldsymbol{H_y x} = \mathbf{0} \quad \Longleftrightarrow \quad \boldsymbol{Hx} = \mathbf{0}. \tag{1}$$

More generally, if $\boldsymbol{s_y} = \boldsymbol{A_y s}$, *then*

$$\boldsymbol{H_y x} = \boldsymbol{s_y} \quad \Longleftrightarrow \quad \boldsymbol{Hx} = \boldsymbol{s}. \tag{2}$$

Proof. The matrix $\boldsymbol{A_y}$ is invertible. Hence $\boldsymbol{A_y Hx} = \mathbf{0}$ holds if and only if $\boldsymbol{Hx} = \mathbf{0}$. Likewise, $\boldsymbol{A_y Hx} = \boldsymbol{A_y s}$ holds if and only if $\boldsymbol{Hx} = \boldsymbol{s}$, after multiplying both sides by $\boldsymbol{A_y}^{-1}$. □

Taking $\boldsymbol{s} = \boldsymbol{H y}_b$ gives $\boldsymbol{s_y} = \boldsymbol{H_y y}_b$, so the transformed syndrome and transformed graph remain exactly consistent. Finally, the construction uses row operations only. No column is exchanged or permuted, and output coordinate i continues to refer to the original codeword coordinate i. What changes is the computational presentation of the constraints, not the code, received word, or decoding target.

B Algorithm and Implementation Details

B.1 One-Pass Construction

The complete preprocessor for one received word is given below. It is deterministic under the stable tie rule and returns both objects required by a syndrome-based base decoder.

Input: full-row-rank $\boldsymbol{H} \in \{0,1\}^{(n-k)\times n}$ and $\boldsymbol{y} \in \mathbb{R}^n$.

Output: receiver-conditioned $(\boldsymbol{H_y}, \boldsymbol{s_y})$ and selected basis $\mathcal{B}_{\boldsymbol{y}}$.

1. Form the hard decision $\boldsymbol{y}_b$ and the original syndrome $\boldsymbol{s} = \boldsymbol{H y}_b$ over $\mathrm{GF}(2)$.
2. Stably sort coordinate indices by increasing $|y_i|$; break equal-magnitude ties by the original coordinate index.
3. Initialize $\mathcal{B}_{\boldsymbol{y}}$ as an empty ordered list. Scan the sorted indices and append i to $\mathcal{B}_{\boldsymbol{y}}$ only if $\mathrm{rank}([\boldsymbol{H}[:, \mathcal{B}_{\boldsymbol{y}}]\ \boldsymbol{H}[:, i]]) = |\mathcal{B}_{\boldsymbol{y}}| + 1$. Stop when $|\mathcal{B}_{\boldsymbol{y}}| = n - k$.
4. Set $\boldsymbol{M_y} = \boldsymbol{H}[:, \mathcal{B}_{\boldsymbol{y}}]$ and use row elimination over $\mathrm{GF}(2)$ to obtain $\boldsymbol{A_y M_y} = \boldsymbol{I}_{n-k}$.
5. Return $\boldsymbol{H_y} = \boldsymbol{A_y H}$ and $\boldsymbol{s_y} = \boldsymbol{A_y s}$.

Incremental elimination maintains the pivots of the currently selected columns, so a candidate does not require a fresh rank computation from scratch. The same elimination can accumulate the row transform and avoid a separate explicit inverse. With a direct dense implementation, the total work is $\mathcal{O}(n(n-k)^2)$ bit operations after the $\mathcal{O}(n \log n)$ reliability sort. In practice, the binary rows are bit-packed and all words in a batch are processed in parallel.

B.2 Base-Decoder Interface

The preprocessor is called once for each received word before the first neural layer. ECCT builds its standard attention mask from $\boldsymbol{H_y}$ and uses $\boldsymbol{s_y}$ as the syndrome tokens. CrossMPT uses $\boldsymbol{H_y}$ and $\boldsymbol{H_y}^{\mathsf{T}}$ in its two existing directed masks. EfficientMPT substitutes $\boldsymbol{H_y}$ in its parity-check projection and aggregation operators. In all three cases, $|\boldsymbol{y}|$, token order, decoder depth, feature dimension, prediction head, loss, and target remain unchanged.

C Additional Results

Figures 6 and 7 report the EfficientMPT BER and BLER curves on the nine main-paper codes. Tables 4–6 provide the 4–6 dB numerical results for all three base decoders.

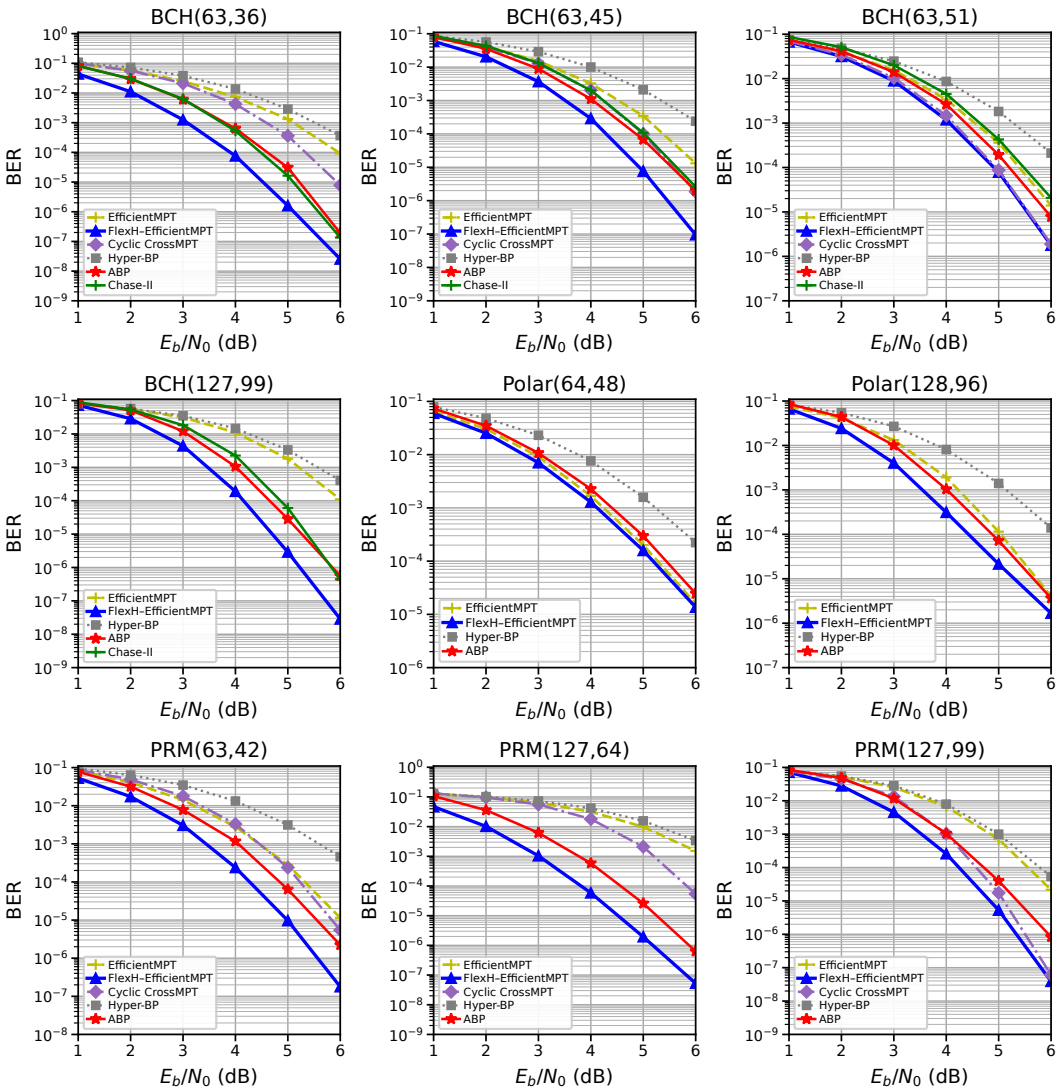


Figure 6: BER on the nine main-paper codes for EfficientMPT, FlexH–EfficientMPT, Cyclic CrossMPT, Hyper-BP, ABP, and Chase-II for BCH.

The results show consistent BER and BLER improvements over the corresponding ECCT, CrossMPT, and EfficientMPT baselines. The gains are largest on BCH and PRM, while Polar retains smaller but consistent improvements. FlexH–EfficientMPT outperforms Hyper-BP throughout the tested range and surpasses ABP and Chase-II in nearly all matched comparisons, with BLER close to the available ML references at several BCH and Polar operating points. The comparison with Cyclic CrossMPT further supports the BER gains beyond standard fixed-graph baselines: on PRM(127,64) at 6 dB, FlexH–CrossMPT reduces BER by more than three orders of magnitude relative to this baseline. Tables 4–6 give the detailed comparisons for each neural architecture.

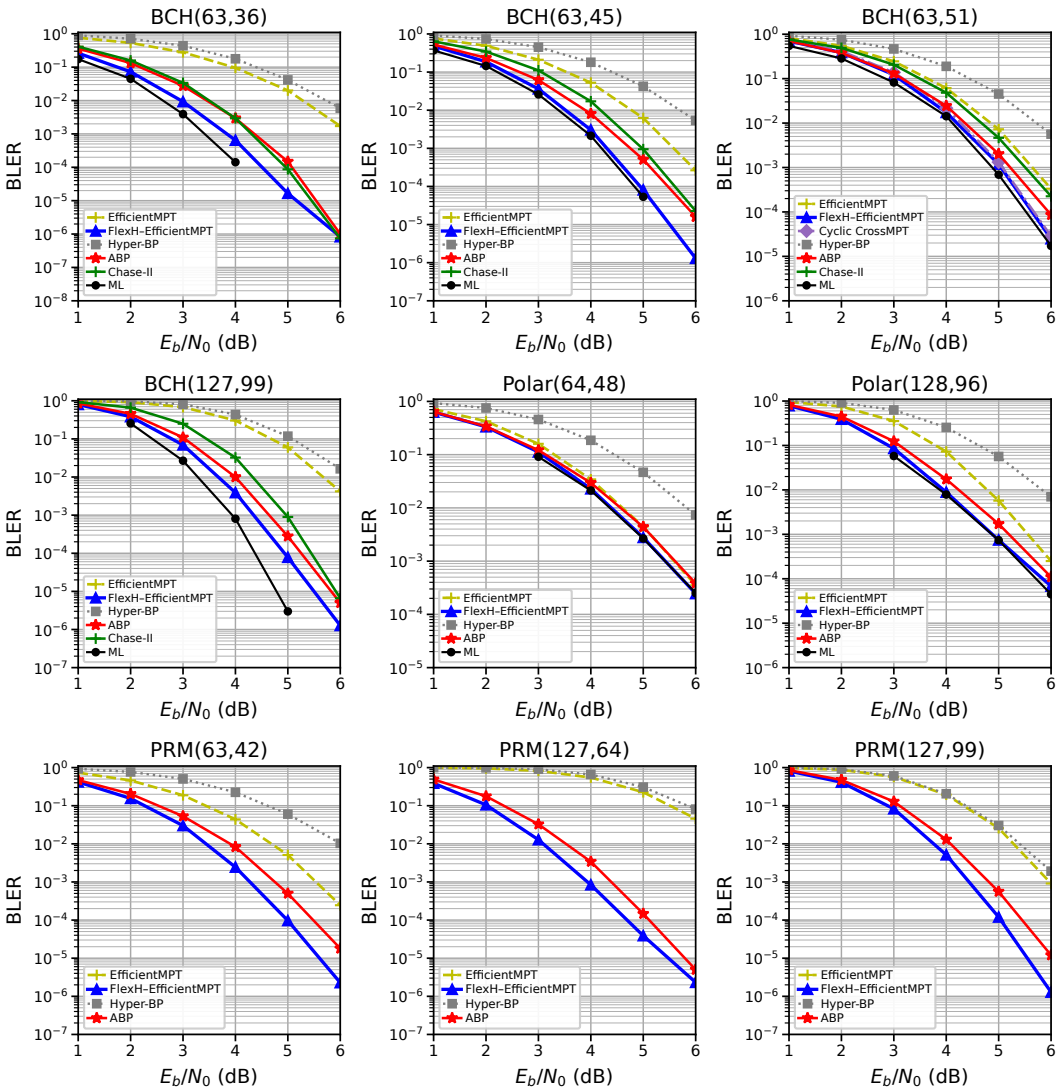


Figure 7: BLER on the nine main-paper codes for EfficientMPT, FlexH–EfficientMPT, Hyper-BP, ABP, and Chase-II for BCH, together with the available Cyclic CrossMPT and ML references.

Table 4: Complete EfficientMPT comparison across the nine main codes at 4–6 dB. Each entry is BER / BLER. The comparison includes Cyclic CrossMPT (Xiao et al., 2026), Hyper-BP, ABP, Chase-II for BCH, and ML BLER references. Unavailable values are marked by a dash. The best available BER and BLER at each SNR are bold.

Code	Decoder	4 dB	5 dB	6 dB
BCH(63,36)	EfficientMPT	7.31×10^{-3} / 9.56×10^{-2}	1.32×10^{-3} / 2.00×10^{-2}	9.21×10^{-5} / 1.71×10^{-3}
	FlexH–EfficientMPT	$\mathbf{7.58\times10^{-5}}$ / 6.47×10^{-4}	$\mathbf{1.59\times10^{-6}}$ / $\mathbf{1.65\times10^{-5}}$	$\mathbf{2.54\times10^{-8}}$ / 8.29×10^{-7}
	Cyclic CrossMPT	4.23×10^{-3} / – –	3.61×10^{-4} / – –	7.84×10^{-6} / – –
	Hyper-BP	1.36×10^{-2} / 1.78×10^{-1}	2.86×10^{-3} / 4.14×10^{-2}	3.67×10^{-4} / 5.90×10^{-3}
	ABP	6.33×10^{-4} / 2.94×10^{-3}	3.07×10^{-5} / 1.48×10^{-4}	1.90×10^{-7} / 1.00×10^{-6}
	Chase-II	5.30×10^{-4} / 2.93×10^{-3}	1.64×10^{-5} / 8.71×10^{-5}	1.35×10^{-7} / $\mathbf{7.40\times10^{-7}}$
	ML	– – / $\mathbf{1.42\times10^{-4}}$	– – / – –	– – / – –

Continued on next page

Table 4 continued

Code	Decoder	4 dB	5 dB	6 dB
BCH(63,45)	EfficientMPT	3.27×10^{-3} / 5.27×10^{-2}	3.44×10^{-4} / 6.22×10^{-3}	1.32×10^{-5} / 2.65×10^{-4}
	FlexH–EfficientMPT	$\mathbf{2.89\times10^{-4}}$ / 3.03×10^{-3}	$\mathbf{7.65\times10^{-6}}$ / 8.10×10^{-5}	$\mathbf{9.47\times10^{-8}}$ / $\mathbf{1.30\times10^{-6}}$
	Cyclic CrossMPT	2.04×10^{-3} / – –	9.98×10^{-5} / – –	1.86×10^{-6} / – –
	Hyper-BP	1.01×10^{-2} / 1.81×10^{-1}	2.10×10^{-3} / 4.23×10^{-2}	2.40×10^{-4} / 5.39×10^{-3}
	ABP	1.11×10^{-3} / 8.05×10^{-3}	6.80×10^{-5} / 5.07×10^{-4}	2.00×10^{-6} / 1.60×10^{-5}
	Chase-II	2.02×10^{-3} / 1.71×10^{-2}	1.08×10^{-4} / 9.48×10^{-4}	2.60×10^{-6} / 2.27×10^{-5}
	ML	– – / $\mathbf{2.13\times10^{-3}}$	– – / $\mathbf{5.34\times10^{-5}}$	– – / – –
BCH(63,51)	EfficientMPT	3.42×10^{-3} / 6.11×10^{-2}	3.61×10^{-4} / 7.25×10^{-3}	1.39×10^{-5} / 3.33×10^{-4}
	FlexH–EfficientMPT	$\mathbf{1.19\times10^{-3}}$ / 1.73×10^{-2}	$\mathbf{7.85\times10^{-5}}$ / 1.19×10^{-3}	$\mathbf{1.78\times10^{-6}}$ / 2.48×10^{-5}
	Cyclic CrossMPT	1.47×10^{-3} / 2.17×10^{-2}	8.73×10^{-5} / 1.32×10^{-3}	1.90×10^{-6} / 2.86×10^{-5}
	Hyper-BP	8.63×10^{-3} / 1.89×10^{-1}	1.83×10^{-3} / 4.51×10^{-2}	2.11×10^{-4} / 5.64×10^{-3}
	ABP	2.65×10^{-3} / 2.44×10^{-2}	1.92×10^{-4} / 2.01×10^{-3}	7.76×10^{-6} / 8.60×10^{-5}
	Chase-II	4.53×10^{-3} / 4.82×10^{-2}	4.29×10^{-4} / 4.65×10^{-3}	2.08×10^{-5} / 2.25×10^{-4}
	ML	– – / $\mathbf{1.43\times10^{-2}}$	– – / $\mathbf{6.89\times10^{-4}}$	– – / $\mathbf{1.72\times10^{-5}}$
BCH(127,99)	EfficientMPT	1.11×10^{-2} / 3.04×10^{-1}	1.82×10^{-3} / 5.98×10^{-2}	1.10×10^{-4} / 4.21×10^{-3}
	FlexH–EfficientMPT	$\mathbf{1.91\times10^{-4}}$ / 3.89×10^{-3}	$\mathbf{2.88\times10^{-6}}$ / 7.80×10^{-5}	$\mathbf{2.85\times10^{-8}}$ / $\mathbf{1.27\times10^{-6}}$
	Hyper-BP	1.47×10^{-2} / 4.36×10^{-1}	3.33×10^{-3} / 1.18×10^{-1}	4.04×10^{-4} / 1.64×10^{-2}
	ABP	1.06×10^{-3} / 9.99×10^{-3}	2.84×10^{-5} / 2.79×10^{-4}	5.59×10^{-7} / 5.00×10^{-6}
	Chase-II	2.25×10^{-3} / 3.24×10^{-2}	6.02×10^{-5} / 8.97×10^{-4}	4.44×10^{-7} / 6.63×10^{-6}
	ML	– – / $\mathbf{8.07\times10^{-4}}$	– – / $\mathbf{2.98\times10^{-6}}$	– – / – –
Polar(64,48)	EfficientMPT	1.68×10^{-3} / 3.43×10^{-2}	2.00×10^{-4} / 4.46×10^{-3}	1.60×10^{-5} / 3.53×10^{-4}
	FlexH–EfficientMPT	$\mathbf{1.28\times10^{-3}}$ / 2.28×10^{-2}	$\mathbf{1.58\times10^{-4}}$ / 2.78×10^{-3}	$\mathbf{1.36\times10^{-5}}$ / $\mathbf{2.45\times10^{-4}}$
	Hyper-BP	7.63×10^{-3} / 1.86×10^{-1}	1.58×10^{-3} / 4.66×10^{-2}	2.23×10^{-4} / 7.37×10^{-3}
	ABP	2.27×10^{-3} / 2.95×10^{-2}	2.98×10^{-4} / 4.39×10^{-3}	2.47×10^{-5} / 3.83×10^{-4}
	ML	– – / $\mathbf{2.12\times10^{-2}}$	– – / $\mathbf{2.66\times10^{-3}}$	– – / 2.51×10^{-4}
Polar(128,96)	EfficientMPT	1.91×10^{-3} / 7.28×10^{-2}	1.15×10^{-4} / 5.73×10^{-3}	4.07×10^{-6} / 2.50×10^{-4}
	FlexH–EfficientMPT	$\mathbf{3.12\times10^{-4}}$ / 8.88×10^{-3}	$\mathbf{2.14\times10^{-5}}$ / 7.43×10^{-4}	$\mathbf{1.67\times10^{-6}}$ / 6.82×10^{-5}
	Hyper-BP	8.08×10^{-3} / 2.55×10^{-1}	1.40×10^{-3} / 5.66×10^{-2}	1.40×10^{-4} / 7.01×10^{-3}
	ABP	1.04×10^{-3} / 1.74×10^{-2}	7.20×10^{-5} / 1.71×10^{-3}	3.65×10^{-6} / 1.09×10^{-4}
	ML	– – / $\mathbf{7.85\times10^{-3}}$	– – / $\mathbf{7.36\times10^{-4}}$	– – / $\mathbf{4.48\times10^{-5}}$
PRM(63,42)	EfficientMPT	2.82×10^{-3} / 4.39×10^{-2}	2.82×10^{-4} / 5.09×10^{-3}	1.15×10^{-5} / 2.45×10^{-4}
	FlexH–EfficientMPT	$\mathbf{2.39\times10^{-4}}$ / $\mathbf{2.44\times10^{-3}}$	$\mathbf{9.67\times10^{-6}}$ / $\mathbf{9.69\times10^{-5}}$	$\mathbf{1.78\times10^{-7}}$ / $\mathbf{2.28\times10^{-6}}$
	Cyclic CrossMPT	3.31×10^{-3} / – –	2.39×10^{-4} / – –	5.47×10^{-6} / – –
	Hyper-BP	1.33×10^{-2} / 2.25×10^{-1}	3.13×10^{-3} / 6.02×10^{-2}	4.61×10^{-4} / 1.03×10^{-2}
	ABP	1.17×10^{-3} / 8.31×10^{-3}	6.39×10^{-5} / 4.98×10^{-4}	2.22×10^{-6} / 1.80×10^{-5}
PRM(127,64)	EfficientMPT	3.18×10^{-2} / 5.48×10^{-1}	9.82×10^{-3} / 2.26×10^{-1}	1.52×10^{-3} / 4.58×10^{-2}
	FlexH–EfficientMPT	$\mathbf{5.86\times10^{-5}}$ / $\mathbf{8.40\times10^{-4}}$	$\mathbf{1.94\times10^{-6}}$ / $\mathbf{3.89\times10^{-5}}$	$\mathbf{5.26\times10^{-8}}$ / $\mathbf{2.32\times10^{-6}}$
	Cyclic CrossMPT	1.80×10^{-2} / – –	2.10×10^{-3} / – –	5.41×10^{-5} / – –
	Hyper-BP	4.19×10^{-2} / 6.60×10^{-1}	1.56×10^{-2} / 3.09×10^{-1}	3.38×10^{-3} / 8.14×10^{-2}
	ABP	5.86×10^{-4} / 3.41×10^{-3}	2.60×10^{-5} / 1.47×10^{-4}	6.30×10^{-7} / 5.00×10^{-6}
PRM(127,99)	EfficientMPT	6.68×10^{-3} / 1.95×10^{-1}	7.07×10^{-4} / 2.58×10^{-2}	2.16×10^{-5} / 9.13×10^{-4}
	FlexH–EfficientMPT	$\mathbf{2.58\times10^{-4}}$ / $\mathbf{5.13\times10^{-3}}$	$\mathbf{5.28\times10^{-6}}$ / $\mathbf{1.20\times10^{-4}}$	$\mathbf{3.88\times10^{-8}}$ / $\mathbf{1.29\times10^{-6}}$
	Cyclic CrossMPT	1.04×10^{-3} / – –	1.74×10^{-5} / – –	6.14×10^{-8} / – –
	Hyper-BP	7.93×10^{-3} / 2.06×10^{-1}	1.00×10^{-3} / 3.02×10^{-2}	5.36×10^{-5} / 1.92×10^{-3}
	ABP	1.04×10^{-3} / 1.30×10^{-2}	3.98×10^{-5} / 5.58×10^{-4}	8.35×10^{-7} / 1.20×10^{-5}

Table 5: Complete ECCT comparison across the nine main codes at 4–6 dB. Each entry is BER / BLER. The comparison includes Cyclic CrossMPT (Xiao et al., 2026), Hyper-BP, ABP, Chase-II for BCH, and ML BLER references. Unavailable values are marked by a dash. The best available BER and BLER at each SNR are bold.

Code	Decoder	4 dB	5 dB	6 dB
BCH(63,36)	ECCT	5.62×10^{-3} / 7.47×10^{-2}	9.05×10^{-4} / 1.47×10^{-2}	6.67×10^{-5} / 1.34×10^{-3}
	FlexH–ECCT	$\mathbf{1.31\times10^{-4}}$ / 1.48×10^{-3}	$\mathbf{3.76\times10^{-6}}$ / $\mathbf{6.56\times10^{-5}}$	$\mathbf{7.63\times10^{-8}}$ / 2.46×10^{-6}
	Cyclic CrossMPT	4.23×10^{-3} / – –	3.61×10^{-4} / – –	7.84×10^{-6} / – –
	Hyper-BP	1.36×10^{-2} / 1.78×10^{-1}	2.86×10^{-3} / 4.14×10^{-2}	3.67×10^{-4} / 5.90×10^{-3}
	ABP	6.33×10^{-4} / 2.94×10^{-3}	3.07×10^{-5} / 1.48×10^{-4}	1.90×10^{-7} / 1.00×10^{-6}
	Chase-II	5.30×10^{-4} / 2.93×10^{-3}	1.64×10^{-5} / 8.71×10^{-5}	1.35×10^{-7} / $\mathbf{7.40\times10^{-7}}$
	ML	– – / $\mathbf{1.42\times10^{-4}}$	– – / – –	– – / – –

Continued on next page

Table 5 continued

Code	Decoder	4 dB	5 dB	6 dB
BCH(63,45)	ECCT	3.84×10^{-3} / 6.71×10^{-2}	4.48×10^{-4} / 9.17×10^{-3}	2.38×10^{-5} / 5.60×10^{-4}
	FlexH–ECCT	$\mathbf{3.06\times10^{-4}}$ / 3.39×10^{-3}	$\mathbf{8.59\times10^{-6}}$ / 1.14×10^{-4}	$\mathbf{1.05\times10^{-7}}$ / $\mathbf{1.85\times10^{-6}}$
	Cyclic CrossMPT	2.04×10^{-3} / – –	9.98×10^{-5} / – –	1.86×10^{-6} / – –
	Hyper-BP	1.01×10^{-2} / 1.81×10^{-1}	2.10×10^{-3} / 4.23×10^{-2}	2.40×10^{-4} / 5.39×10^{-3}
	ABP	1.11×10^{-3} / 8.05×10^{-3}	6.80×10^{-5} / 5.07×10^{-4}	2.00×10^{-6} / 1.60×10^{-5}
	Chase-II	2.02×10^{-3} / 1.71×10^{-2}	1.08×10^{-4} / 9.48×10^{-4}	2.60×10^{-6} / 2.27×10^{-5}
	ML	– – / $\mathbf{2.13\times10^{-3}}$	– – / $\mathbf{5.34\times10^{-5}}$	– – / – –
BCH(63,51)	ECCT	3.53×10^{-3} / 6.47×10^{-2}	3.96×10^{-4} / 8.05×10^{-3}	1.78×10^{-5} / 4.40×10^{-4}
	FlexH–ECCT	$\mathbf{1.18\times10^{-3}}$ / 1.71×10^{-2}	$\mathbf{7.88\times10^{-5}}$ / 1.22×10^{-3}	$\mathbf{1.73\times10^{-6}}$ / 2.36×10^{-5}
	Cyclic CrossMPT	1.47×10^{-3} / 2.17×10^{-2}	8.73×10^{-5} / 1.32×10^{-3}	1.90×10^{-6} / 2.86×10^{-5}
	Hyper-BP	8.63×10^{-3} / 1.89×10^{-1}	1.83×10^{-3} / 4.51×10^{-2}	2.11×10^{-4} / 5.64×10^{-3}
	ABP	2.65×10^{-3} / 2.44×10^{-2}	1.92×10^{-4} / 2.01×10^{-3}	7.76×10^{-6} / 8.60×10^{-5}
	Chase-II	4.53×10^{-3} / 4.82×10^{-2}	4.29×10^{-4} / 4.65×10^{-3}	2.08×10^{-5} / 2.25×10^{-4}
	ML	– – / $\mathbf{1.43\times10^{-2}}$	– – / $\mathbf{6.89\times10^{-4}}$	– – / $\mathbf{1.72\times10^{-5}}$
BCH(127,99)	ECCT	1.05×10^{-2} / 2.92×10^{-1}	1.61×10^{-3} / 5.47×10^{-2}	8.95×10^{-5} / 3.55×10^{-3}
	FlexH–ECCT	$\mathbf{2.38\times10^{-4}}$ / 5.44×10^{-3}	$\mathbf{5.51\times10^{-6}}$ / 1.68×10^{-4}	$\mathbf{4.86\times10^{-8}}$ / $\mathbf{2.43\times10^{-6}}$
	Hyper-BP	1.47×10^{-2} / 4.36×10^{-1}	3.33×10^{-3} / 1.18×10^{-1}	4.04×10^{-4} / 1.64×10^{-2}
	ABP	1.06×10^{-3} / 9.99×10^{-3}	2.84×10^{-5} / 2.79×10^{-4}	5.59×10^{-7} / 5.00×10^{-6}
	Chase-II	2.25×10^{-3} / 3.24×10^{-2}	6.02×10^{-5} / 8.97×10^{-4}	4.44×10^{-7} / 6.63×10^{-6}
	ML	– – / $\mathbf{8.07\times10^{-4}}$	– – / $\mathbf{2.98\times10^{-6}}$	– – / – –
Polar(64,48)	ECCT	1.69×10^{-3} / 3.80×10^{-2}	1.98×10^{-4} / 5.01×10^{-3}	1.78×10^{-5} / 4.33×10^{-4}
	FlexH–ECCT	$\mathbf{1.29\times10^{-3}}$ / 2.27×10^{-2}	$\mathbf{1.59\times10^{-4}}$ / 2.79×10^{-3}	$\mathbf{1.40\times10^{-5}}$ / $\mathbf{2.44\times10^{-4}}$
	Hyper-BP	7.63×10^{-3} / 1.86×10^{-1}	1.58×10^{-3} / 4.66×10^{-2}	2.23×10^{-4} / 7.37×10^{-3}
	ABP	2.27×10^{-3} / 2.95×10^{-2}	2.98×10^{-4} / 4.39×10^{-3}	2.47×10^{-5} / 3.83×10^{-4}
	ML	– – / $\mathbf{2.12\times10^{-2}}$	– – / $\mathbf{2.66\times10^{-3}}$	– – / 2.51×10^{-4}
Polar(128,96)	ECCT	1.74×10^{-3} / 7.40×10^{-2}	1.02×10^{-4} / 5.71×10^{-3}	3.60×10^{-6} / 2.27×10^{-4}
	FlexH–ECCT	$\mathbf{3.31\times10^{-4}}$ / 9.58×10^{-3}	$\mathbf{2.26\times10^{-5}}$ / 7.40×10^{-4}	$\mathbf{1.62\times10^{-6}}$ / 6.04×10^{-5}
	Hyper-BP	8.08×10^{-3} / 2.55×10^{-1}	1.40×10^{-3} / 5.66×10^{-2}	1.40×10^{-4} / 7.01×10^{-3}
	ABP	1.04×10^{-3} / 1.74×10^{-2}	7.20×10^{-5} / 1.71×10^{-3}	3.65×10^{-6} / 1.09×10^{-4}
	ML	– – / $\mathbf{7.85\times10^{-3}}$	– – / $\mathbf{7.36\times10^{-4}}$	– – / $\mathbf{4.48\times10^{-5}}$
PRM(63,42)	ECCT	1.86×10^{-3} / 3.28×10^{-2}	1.40×10^{-4} / 3.08×10^{-3}	4.37×10^{-6} / 1.13×10^{-4}
	FlexH–ECCT	$\mathbf{2.56\times10^{-4}}$ / $\mathbf{2.71\times10^{-3}}$	$\mathbf{1.03\times10^{-5}}$ / $\mathbf{1.18\times10^{-4}}$	$\mathbf{1.98\times10^{-7}}$ / $\mathbf{2.92\times10^{-6}}$
	Cyclic CrossMPT	3.31×10^{-3} / – –	2.39×10^{-4} / – –	5.47×10^{-6} / – –
	Hyper-BP	1.33×10^{-2} / 2.25×10^{-1}	3.13×10^{-3} / 6.02×10^{-2}	4.61×10^{-4} / 1.03×10^{-2}
	ABP	1.17×10^{-3} / 8.31×10^{-3}	6.39×10^{-5} / 4.98×10^{-4}	2.22×10^{-6} / 1.80×10^{-5}
PRM(127,64)	ECCT	2.39×10^{-2} / 4.31×10^{-1}	6.04×10^{-3} / 1.51×10^{-1}	7.35×10^{-4} / 2.57×10^{-2}
	FlexH–ECCT	$\mathbf{6.24\times10^{-5}}$ / $\mathbf{1.11\times10^{-3}}$	$\mathbf{2.09\times10^{-6}}$ / $\mathbf{5.00\times10^{-5}}$	$\mathbf{5.23\times10^{-8}}$ / $\mathbf{3.71\times10^{-6}}$
	Cyclic CrossMPT	1.80×10^{-2} / – –	2.10×10^{-3} / – –	5.41×10^{-5} / – –
	Hyper-BP	4.19×10^{-2} / 6.60×10^{-1}	1.56×10^{-2} / 3.09×10^{-1}	3.38×10^{-3} / 8.14×10^{-2}
	ABP	5.86×10^{-4} / 3.41×10^{-3}	2.60×10^{-5} / 1.47×10^{-4}	6.30×10^{-7} / 5.00×10^{-6}
PRM(127,99)	ECCT	6.99×10^{-3} / 2.22×10^{-1}	7.38×10^{-4} / 3.09×10^{-2}	2.87×10^{-5} / 1.59×10^{-3}
	FlexH–ECCT	$\mathbf{2.82\times10^{-4}}$ / $\mathbf{6.14\times10^{-3}}$	$\mathbf{6.39\times10^{-6}}$ / $\mathbf{1.67\times10^{-4}}$	$\mathbf{5.41\times10^{-8}}$ / $\mathbf{2.10\times10^{-6}}$
	Cyclic CrossMPT	1.04×10^{-3} / – –	1.74×10^{-5} / – –	6.14×10^{-8} / – –
	Hyper-BP	7.93×10^{-3} / 2.06×10^{-1}	1.00×10^{-3} / 3.02×10^{-2}	5.36×10^{-5} / 1.92×10^{-3}
	ABP	1.04×10^{-3} / 1.30×10^{-2}	3.98×10^{-5} / 5.58×10^{-4}	8.35×10^{-7} / 1.20×10^{-5}

Table 6: Complete CrossMPT comparison across the nine main codes at 4–6 dB. Each entry is BER / BLER. The comparison includes Cyclic CrossMPT (Xiao et al., 2026), Hyper-BP, ABP, Chase-II for BCH, and ML BLER references. Unavailable values are marked by a dash. The best available BER and BLER at each SNR are bold.

Code	Decoder	4 dB	5 dB	6 dB
BCH(63,36)	CrossMPT	6.18×10^{-3} / 7.74×10^{-2}	1.03×10^{-3} / 1.48×10^{-2}	6.94×10^{-5} / 1.14×10^{-3}
	FlexH–CrossMPT	$\mathbf{9.56\times10^{-5}}$ / 8.74×10^{-4}	$\mathbf{2.09\times10^{-6}}$ / $\mathbf{3.01\times10^{-5}}$	$\mathbf{4.94\times10^{-8}}$ / 1.78×10^{-6}
	Cyclic CrossMPT	4.23×10^{-3} / – –	3.61×10^{-4} / – –	7.84×10^{-6} / – –
	Hyper-BP	1.36×10^{-2} / 1.78×10^{-1}	2.86×10^{-3} / 4.14×10^{-2}	3.67×10^{-4} / 5.90×10^{-3}
	ABP	6.33×10^{-4} / 2.94×10^{-3}	3.07×10^{-5} / 1.48×10^{-4}	1.90×10^{-7} / 1.00×10^{-6}
	Chase-II	5.30×10^{-4} / 2.93×10^{-3}	1.64×10^{-5} / 8.71×10^{-5}	1.35×10^{-7} / $\mathbf{7.40\times10^{-7}}$
	ML	– – / $\mathbf{1.42\times10^{-4}}$	– – / – –	– – / – –

Continued on next page

Table 6 continued

Code	Decoder	4 dB	5 dB	6 dB
BCH(63,45)	CrossMPT	2.80×10^{-3} / 4.27×10^{-2}	2.79×10^{-4} / 4.80×10^{-3}	1.02×10^{-5} / 1.79×10^{-4}
	FlexH–CrossMPT	$\mathbf{2.92\times10^{-4}}$ / 3.13×10^{-3}	$\mathbf{8.38\times10^{-6}}$ / 1.02×10^{-4}	$\mathbf{1.42\times10^{-7}}$ / $\mathbf{3.21\times10^{-6}}$
	Cyclic CrossMPT	2.04×10^{-3} / – –	9.98×10^{-5} / – –	1.86×10^{-6} / – –
	Hyper-BP	1.01×10^{-2} / 1.81×10^{-1}	2.10×10^{-3} / 4.23×10^{-2}	2.40×10^{-4} / 5.39×10^{-3}
	ABP	1.11×10^{-3} / 8.05×10^{-3}	6.80×10^{-5} / 5.07×10^{-4}	2.00×10^{-6} / 1.60×10^{-5}
	Chase-II	2.02×10^{-3} / 1.71×10^{-2}	1.08×10^{-4} / 9.48×10^{-4}	2.60×10^{-6} / 2.27×10^{-5}
	ML	– – / $\mathbf{2.13\times10^{-3}}$	– – / $\mathbf{5.34\times10^{-5}}$	– – / – –
BCH(63,51)	CrossMPT	3.08×10^{-3} / 5.26×10^{-2}	3.19×10^{-4} / 5.86×10^{-3}	1.21×10^{-5} / 2.52×10^{-4}
	FlexH–CrossMPT	$\mathbf{1.19\times10^{-3}}$ / 1.70×10^{-2}	$\mathbf{7.67\times10^{-5}}$ / 1.15×10^{-3}	$\mathbf{1.77\times10^{-6}}$ / 2.69×10^{-5}
	Cyclic CrossMPT	1.47×10^{-3} / 2.17×10^{-2}	8.73×10^{-5} / 1.32×10^{-3}	1.90×10^{-6} / 2.86×10^{-5}
	Hyper-BP	8.63×10^{-3} / 1.89×10^{-1}	1.83×10^{-3} / 4.51×10^{-2}	2.11×10^{-4} / 5.64×10^{-3}
	ABP	2.65×10^{-3} / 2.44×10^{-2}	1.92×10^{-4} / 2.01×10^{-3}	7.76×10^{-6} / 8.60×10^{-5}
	Chase-II	4.53×10^{-3} / 4.82×10^{-2}	4.29×10^{-4} / 4.65×10^{-3}	2.08×10^{-5} / 2.25×10^{-4}
	ML	– – / $\mathbf{1.43\times10^{-2}}$	– – / $\mathbf{6.89\times10^{-4}}$	– – / $\mathbf{1.72\times10^{-5}}$
BCH(127,99)	CrossMPT	1.01×10^{-2} / 2.72×10^{-1}	1.51×10^{-3} / 4.80×10^{-2}	8.22×10^{-5} / 2.99×10^{-3}
	FlexH–CrossMPT	$\mathbf{1.49\times10^{-4}}$ / 2.91×10^{-3}	$\mathbf{1.71\times10^{-6}}$ / 4.33×10^{-5}	$\mathbf{1.44\times10^{-8}}$ / $\mathbf{8.59\times10^{-7}}$
	Hyper-BP	1.47×10^{-2} / 4.36×10^{-1}	3.33×10^{-3} / 1.18×10^{-1}	4.04×10^{-4} / 1.64×10^{-2}
	ABP	1.06×10^{-3} / 9.99×10^{-3}	2.84×10^{-5} / 2.79×10^{-4}	5.59×10^{-7} / 5.00×10^{-6}
	Chase-II	2.25×10^{-3} / 3.24×10^{-2}	6.02×10^{-5} / 8.97×10^{-4}	4.44×10^{-7} / 6.63×10^{-6}
	ML	– – / $\mathbf{8.07\times10^{-4}}$	– – / $\mathbf{2.98\times10^{-6}}$	– – / – –
Polar(64,48)	CrossMPT	1.49×10^{-3} / 2.84×10^{-2}	1.68×10^{-4} / 3.47×10^{-3}	1.47×10^{-5} / 2.90×10^{-4}
	FlexH–CrossMPT	$\mathbf{1.29\times10^{-3}}$ / 2.33×10^{-2}	$\mathbf{1.57\times10^{-4}}$ / 2.88×10^{-3}	$\mathbf{1.38\times10^{-5}}$ / $\mathbf{2.42\times10^{-4}}$
	Hyper-BP	7.63×10^{-3} / 1.86×10^{-1}	1.58×10^{-3} / 4.66×10^{-2}	2.23×10^{-4} / 7.37×10^{-3}
	ABP	2.27×10^{-3} / 2.95×10^{-2}	2.98×10^{-4} / 4.39×10^{-3}	2.47×10^{-5} / 3.83×10^{-4}
	ML	– – / $\mathbf{2.12\times10^{-2}}$	– – / $\mathbf{2.66\times10^{-3}}$	– – / 2.51×10^{-4}
Polar(128,96)	CrossMPT	1.39×10^{-3} / 5.24×10^{-2}	7.33×10^{-5} / 3.59×10^{-3}	2.76×10^{-6} / 1.68×10^{-4}
	FlexH–CrossMPT	$\mathbf{3.09\times10^{-4}}$ / 8.56×10^{-3}	$\mathbf{2.23\times10^{-5}}$ / $\mathbf{7.29\times10^{-4}}$	$\mathbf{1.70\times10^{-6}}$ / 5.99×10^{-5}
	Hyper-BP	8.08×10^{-3} / 2.55×10^{-1}	1.40×10^{-3} / 5.66×10^{-2}	1.40×10^{-4} / 7.01×10^{-3}
	ABP	1.04×10^{-3} / 1.74×10^{-2}	7.20×10^{-5} / 1.71×10^{-3}	3.65×10^{-6} / 1.09×10^{-4}
	ML	– – / $\mathbf{7.85\times10^{-3}}$	– – / 7.36×10^{-4}	– – / $\mathbf{4.48\times10^{-5}}$
PRM(63,42)	CrossMPT	1.81×10^{-3} / 2.55×10^{-2}	1.50×10^{-4} / 2.35×10^{-3}	4.18×10^{-6} / 7.71×10^{-5}
	FlexH–CrossMPT	$\mathbf{2.41\times10^{-4}}$ / $\mathbf{2.59\times10^{-3}}$	$\mathbf{8.88\times10^{-6}}$ / $\mathbf{9.73\times10^{-5}}$	$\mathbf{1.93\times10^{-7}}$ / $\mathbf{4.63\times10^{-6}}$
	Cyclic CrossMPT	3.31×10^{-3} / – –	2.39×10^{-4} / – –	5.47×10^{-6} / – –
	Hyper-BP	1.33×10^{-2} / 2.25×10^{-1}	3.13×10^{-3} / 6.02×10^{-2}	4.61×10^{-4} / 1.03×10^{-2}
	ABP	1.17×10^{-3} / 8.31×10^{-3}	6.39×10^{-5} / 4.98×10^{-4}	2.22×10^{-6} / 1.80×10^{-5}
PRM(127,64)	CrossMPT	2.34×10^{-2} / 3.84×10^{-1}	5.63×10^{-3} / 1.20×10^{-1}	6.21×10^{-4} / 1.66×10^{-2}
	FlexH–CrossMPT	$\mathbf{3.61\times10^{-5}}$ / $\mathbf{4.49\times10^{-4}}$	$\mathbf{7.77\times10^{-7}}$ / $\mathbf{1.52\times10^{-5}}$	$\mathbf{2.19\times10^{-8}}$ / $\mathbf{1.76\times10^{-6}}$
	Cyclic CrossMPT	1.80×10^{-2} / – –	2.10×10^{-3} / – –	5.41×10^{-5} / – –
	Hyper-BP	4.19×10^{-2} / 6.60×10^{-1}	1.56×10^{-2} / 3.09×10^{-1}	3.38×10^{-3} / 8.14×10^{-2}
	ABP	5.86×10^{-4} / 3.41×10^{-3}	2.60×10^{-5} / 1.47×10^{-4}	6.30×10^{-7} / 5.00×10^{-6}
PRM(127,99)	CrossMPT	5.60×10^{-3} / 1.55×10^{-1}	5.00×10^{-4} / 1.71×10^{-2}	1.34×10^{-5} / 5.41×10^{-4}
	FlexH–CrossMPT	$\mathbf{2.36\times10^{-4}}$ / $\mathbf{4.81\times10^{-3}}$	$\mathbf{4.94\times10^{-6}}$ / $\mathbf{1.08\times10^{-4}}$	$\mathbf{4.00\times10^{-8}}$ / $\mathbf{1.28\times10^{-6}}$
	Cyclic CrossMPT	1.04×10^{-3} / – –	1.74×10^{-5} / – –	6.14×10^{-8} / – –
	Hyper-BP	7.93×10^{-3} / 2.06×10^{-1}	1.00×10^{-3} / 3.02×10^{-2}	5.36×10^{-5} / 1.92×10^{-3}
	ABP	1.04×10^{-3} / 1.30×10^{-2}	3.98×10^{-5} / 5.58×10^{-4}	8.35×10^{-7} / 1.20×10^{-5}

D Additional and Long-Block Results

D.1 Additional and Long BCH Codes

To assess applicability beyond the main experiments, we also evaluate longer BCH codes, including four with $n > 500$. On these long codes, the original EfficientMPT largely loses its practical error-correction ability: its BER and BLER remain close to a regime with little successful codeword recovery, and its BLER is near one in several cases. FlexH–EfficientMPT restores effective decoding on the same architecture. At 6 dB, BCH(511,385) changes from an EfficientMPT BLER of 9.74×10^{-1} to 2.42×10^{-5} with FlexH–EfficientMPT, while BCH(1023,993) changes from 9.39×10^{-1} to 1.03×10^{-2}. Table 7 also reports BCH(255,239) and the available ML references.

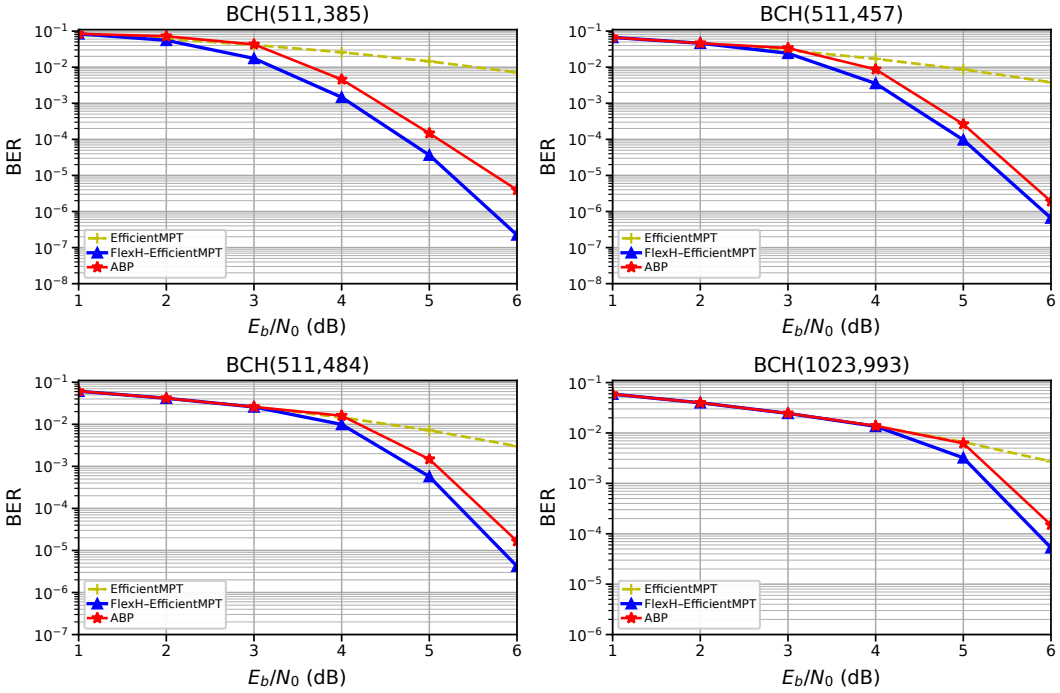


Figure 8: BER on the four BCH codes with $n > 500$ for EfficientMPT, FlexH–EfficientMPT, and ABP.

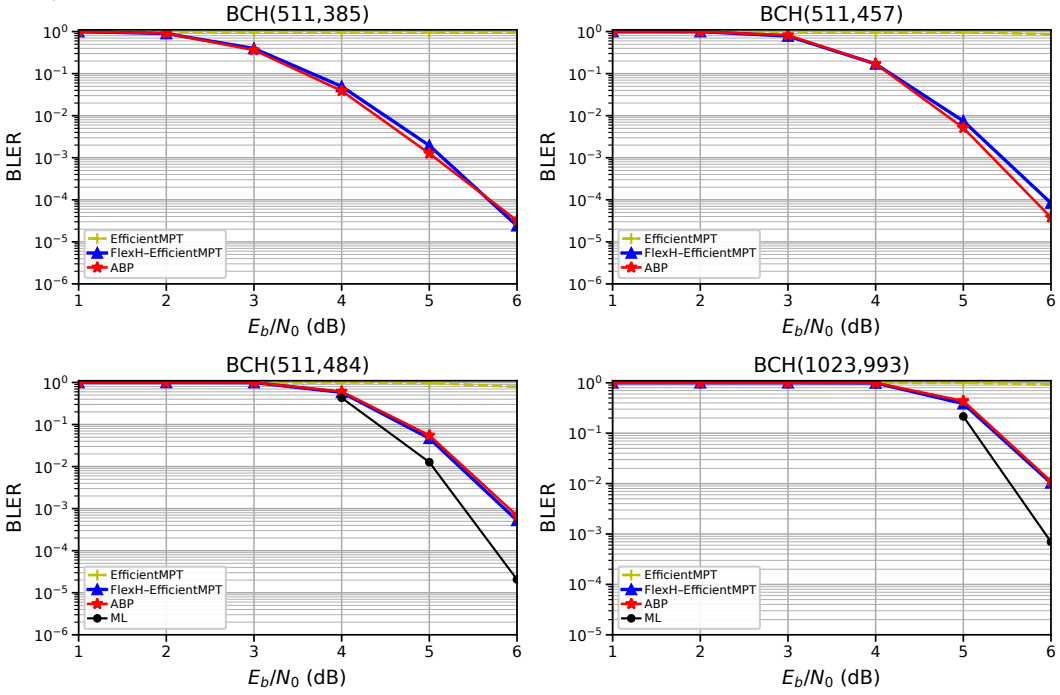


Figure 9: BLER on the four BCH codes with $n > 500$ for EfficientMPT, FlexH–EfficientMPT, ABP, and the available ML references.

Table 7: Additional and long-BCH results at 4–6 dB. Each entry is BER / BLER. ABP is included; ML rows contain BLER only, and a dash denotes an unavailable point. The best available BER and BLER at each SNR are bold.

Code	Decoder	4 dB	5 dB	6 dB
BCH(255,239)	EfficientMPT	1.28×10^{-2} / 7.15×10^{-1}	3.02×10^{-3} / 2.23×10^{-1}	1.97×10^{-4} / 1.76×10^{-2}
	FlexH–EfficientMPT	$\mathbf{7.29\times10^{-3}}$ / 3.29×10^{-1}	$\mathbf{4.57\times10^{-4}}$ / 2.34×10^{-2}	$\mathbf{6.64\times10^{-6}}$ / 3.80×10^{-4}
	ABP	1.20×10^{-2} / 3.52×10^{-1}	1.34×10^{-3} / 4.40×10^{-2}	2.91×10^{-5} / 1.03×10^{-3}
	ML	– – / $\mathbf{2.60\times10^{-1}}$	– – / $\mathbf{1.26\times10^{-2}}$	– – / $\mathbf{1.71\times10^{-4}}$
BCH(511,385)	EfficientMPT	2.59×10^{-2} / 1.00×10^{0}	1.45×10^{-2} / 9.99×10^{-1}	7.16×10^{-3} / 9.74×10^{-1}
	FlexH–EfficientMPT	$\mathbf{1.46\times10^{-3}}$ / 4.96×10^{-2}	$\mathbf{3.66\times10^{-5}}$ / 1.95×10^{-3}	$\mathbf{2.24\times10^{-7}}$ / $\mathbf{2.42\times10^{-5}}$
	ABP	4.56×10^{-3} / $\mathbf{3.91\times10^{-2}}$	1.47×10^{-4} / $\mathbf{1.26\times10^{-3}}$	3.97×10^{-6} / 3.20×10^{-5}
BCH(511,457)	EfficientMPT	1.70×10^{-2} / 1.00×10^{0}	8.69×10^{-3} / 9.88×10^{-1}	3.81×10^{-3} / 8.59×10^{-1}
	FlexH–EfficientMPT	$\mathbf{3.54\times10^{-3}}$ / $\mathbf{1.70\times10^{-1}}$	$\mathbf{9.68\times10^{-5}}$ / 7.47×10^{-3}	$\mathbf{6.59\times10^{-7}}$ / 8.36×10^{-5}
	ABP	8.72×10^{-3} / 1.71×10^{-1}	2.63×10^{-4} / $\mathbf{5.14\times10^{-3}}$	1.91×10^{-6} / $\mathbf{3.80\times10^{-5}}$
BCH(511,484)	EfficientMPT	1.46×10^{-2} / 1.00×10^{0}	7.18×10^{-3} / 9.75×10^{-1}	3.02×10^{-3} / 7.87×10^{-1}
	FlexH–EfficientMPT	$\mathbf{9.93\times10^{-3}}$ / 5.82×10^{-1}	$\mathbf{5.71\times10^{-4}}$ / 4.71×10^{-2}	$\mathbf{4.21\times10^{-6}}$ / 5.24×10^{-4}
	ABP	1.60×10^{-2} / 6.08×10^{-1}	1.48×10^{-3} / 5.53×10^{-2}	1.67×10^{-5} / 6.82×10^{-4}
	ML	– – / $\mathbf{4.31\times10^{-1}}$	– – / $\mathbf{1.27\times10^{-2}}$	– – / $\mathbf{2.07\times10^{-5}}$
BCH(1023,993)	EfficientMPT	$\mathbf{1.36\times10^{-2}}$ / 1.00×10^{0}	6.62×10^{-3} / 9.99×10^{-1}	2.73×10^{-3} / 9.39×10^{-1}
	FlexH–EfficientMPT	$\mathbf{1.36\times10^{-2}}$ / $\mathbf{9.91\times10^{-1}}$	$\mathbf{3.22\times10^{-3}}$ / 3.86×10^{-1}	$\mathbf{5.28\times10^{-5}}$ / 1.03×10^{-2}
	ABP	1.38×10^{-2} / 1.00×10^{0}	6.24×10^{-3} / 4.35×10^{-1}	1.50×10^{-4} / 1.11×10^{-2}
	ML	– – / – –	– – / $\mathbf{2.15\times10^{-1}}$	– – / $\mathbf{6.98\times10^{-4}}$

D.2 Long Polar Results

We also extend the evaluation to Polar codes of lengths 512 and 1024 to examine applicability at larger block lengths. Table 8 compares EfficientMPT, FlexH–EfficientMPT, and ABP at 4–6 dB. Overall, FlexH–EfficientMPT remains close to EfficientMPT and is slightly better at several operating points: it improves both metrics on Polar(512,256) at 4 and 5 dB, lowers BLER on Polar(512,384) at 4 dB, and lowers BLER on Polar(1024,683) at all three SNRs. More importantly, these results are obtained with a $2d$ FFN, only half the hidden width of the $4d$ baseline, which also reduces the total parameter count and FLOPs. The long-Polar results therefore show that receiver-conditioned representations can retain baseline-level performance, and occasionally improve it, while replacing part of the learned capacity otherwise supplied by a wider FFN. This accuracy–efficiency pattern further supports our interpretation of FlexH as providing conditional structural capacity.

Table 8: Long-Polar results at 4–6 dB. Each entry is BER / BLER; the best available BER and BLER at each SNR are bold.

Code	Decoder	4 dB	5 dB	6 dB
Polar(512,256)	EfficientMPT	1.56×10^{-2} / 6.89×10^{-1}	1.34×10^{-3} / 1.56×10^{-1}	$\mathbf{3.36\times10^{-5}}$ / 7.48×10^{-3}
	FlexH–EfficientMPT	$\mathbf{3.96\times10^{-3}}$ / 3.25×10^{-1}	$\mathbf{3.79\times10^{-4}}$ / 9.37×10^{-2}	3.90×10^{-5} / 1.67×10^{-2}
	ABP	8.67×10^{-3} / $\mathbf{4.06\times10^{-2}}$	1.33×10^{-3} / $\mathbf{7.61\times10^{-3}}$	3.83×10^{-4} / $\mathbf{1.90\times10^{-3}}$
Polar(512,384)	EfficientMPT	5.72×10^{-3} / 5.25×10^{-1}	2.75×10^{-4} / 5.56×10^{-2}	$\mathbf{3.78\times10^{-6}}$ / $\mathbf{1.11\times10^{-3}}$
	FlexH–EfficientMPT	5.80×10^{-3} / 4.85×10^{-1}	3.33×10^{-4} / 5.82×10^{-2}	5.84×10^{-6} / 1.51×10^{-3}
	ABP	$\mathbf{1.99\times10^{-3}}$ / $\mathbf{2.46\times10^{-2}}$	$\mathbf{2.72\times10^{-4}}$ / $\mathbf{5.24\times10^{-3}}$	1.06×10^{-4} / 1.40×10^{-3}
Polar(1024,683)	EfficientMPT	$\mathbf{1.37\times10^{-2}}$ / 9.52×10^{-1}	$\mathbf{1.16\times10^{-3}}$ / 3.27×10^{-1}	$\mathbf{2.03\times10^{-5}}$ / 1.12×10^{-2}
	FlexH–EfficientMPT	1.43×10^{-2} / 9.20×10^{-1}	1.22×10^{-3} / 2.72×10^{-1}	2.42×10^{-5} / 1.11×10^{-2}
	ABP	1.45×10^{-2} / $\mathbf{1.00\times10^{-1}}$	2.43×10^{-3} / $\mathbf{1.80\times10^{-2}}$	4.75×10^{-4} / $\mathbf{3.17\times10^{-3}}$

E Extended Accuracy–Efficiency Results

Figure 10 complements the parameter and FLOP comparisons in Figure 4 with peak inference memory. On the same six codes at 6 dB, FlexH–EfficientMPT achieves lower BLER while using less memory than EfficientMPT.

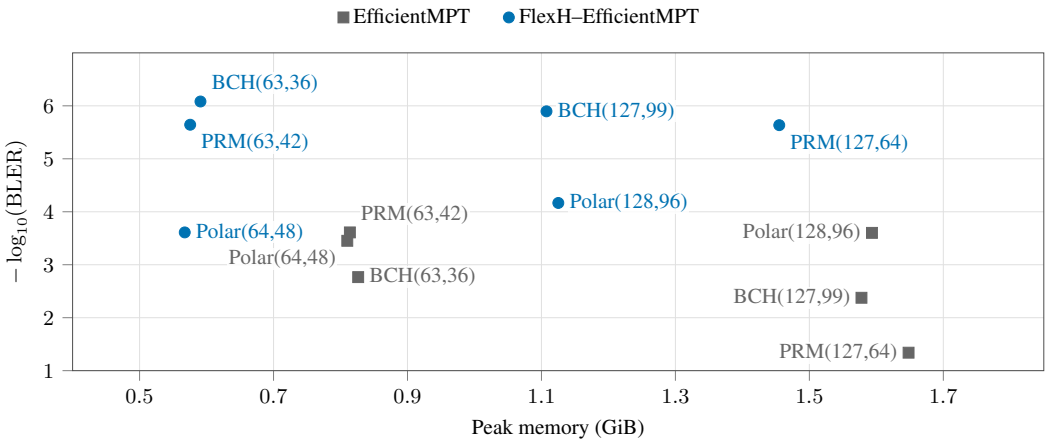


Figure 10: BLER versus peak inference memory at 6 dB, comparing EfficientMPT with FlexH–EfficientMPT on the same six codes as Figure 4.

Table 9 extends the efficiency comparison to five additional BCH cases. FlexH–EfficientMPT uses about 64% of the baseline parameters and 64%–69% of its FLOPs. Peak memory falls to about 68%–76% of the baseline in four cases. The exception is BCH(511,385), where peak memory is about 119% of EfficientMPT despite the lower parameter and FLOP counts.

Table 9: Extended EfficientMPT efficiency results. P: parameters (M); F: FLOPs per codeword (M), including preprocessing for FlexH; M: peak allocated inference memory (GiB) at batch size 2048. Percentages are relative to EfficientMPT (100%). The lower value for each code and metric is bold.

	EfficientMPT			FlexH–EfficientMPT		
Code	P	F	M	P	F	M
BCH(255,239)	1.093 (100%)	600.2 (100%)	3.06 (100%)	**0.698** (64%)	**387.1** (64%)	**2.09** (68%)
BCH(511,385)	1.093 (100%)	1593.1 (100%)	**6.28** (100%)	**0.698** (64%)	**1092.7** (69%)	7.47 (119%)
BCH(511,457)	1.093 (100%)	1314.8 (100%)	6.14 (100%)	**0.698** (64%)	**871.0** (66%)	**4.66** (76%)
BCH(511,484)	1.093 (100%)	1210.5 (100%)	6.09 (100%)	**0.698** (64%)	**787.8** (65%)	**4.21** (69%)
BCH(1023,993)	1.093 (100%)	2381.5 (100%)	12.11 (100%)	**0.698** (64%)	**1554.3** (65%)	**8.37** (69%)

F Extended Ablation Studies

Figure 11 shows the BER ablation curves.

For the Magnitude + syndrome control, the magnitudes within each word are linearly rescaled to $[0, 1]$. The score for each bit subtracts 0.2 times the fraction of its incident checks that are unsatisfied under the base matrix. Columns are scanned in increasing score order, retaining the same rank test as FlexH. Thus, at equal normalized magnitude, a larger fraction of unsatisfied checks gives a bit higher selection priority.

Across the reported 4–6 dB points, FlexH with a $2d$ FFN retains performance close to its $4d$ counterpart. The differences between the two widths are small relative to the gains over Static-H $4d$. Random-full supplies the shuffled equivalent-matrix control. Its occasional small gains show that changing the view need not be destructive, but it does not approach the consistent improvement of

FlexH. In these experiments, assigning identity roles to the most reliable independent coordinates does not reproduce the gains of the least-reliable basis. Finally, adding syndrome information to the magnitude score gives no consistent advantage over the simpler magnitude-only rule.

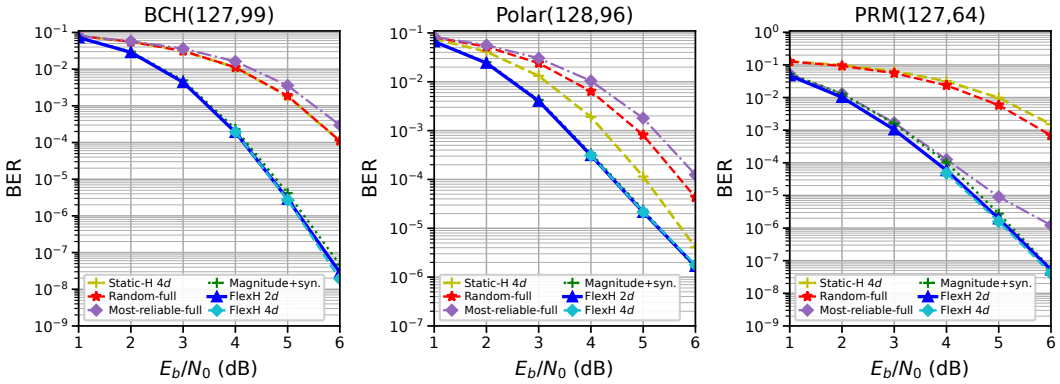


Figure 11: BER curves for the EfficientMPT ablations. Static-H uses the standard $4d$ FFN, the representation controls use $2d$, and FlexH is shown at both $2d$ and $4d$.

F.1 FFN Update Statistics Across Layers

We compare separately trained EfficientMPT checkpoints for Static-H $4d$, FlexH $4d$, and FlexH $2d$ on BCH(127,99), Polar(128,96), and PRM(127,64). At each of 4, 5, and 6 dB, we use 8192 received words with batch size 256, all-zero codewords, and matched noise across configurations using seed 20260802. Figures 5 and 12 report arithmetic means over the three SNRs.

EfficientMPT carries bit features on the variable-node (VN) path and parity-check features on the check-node (CN) path. Let u be the features just before the FFN addition and r_F its output. Figure 5(a) divides mean $\|r_F\|_2$ at initially incorrect bit positions by that at correct positions, defined by channel hard decisions. Figure 5(b) averages $\|r_F\|_2/\|u\|_2$ over tokens; a lower ratio need not mean a smaller absolute output.

For Figure 5(c), hidden unit j has estimated contribution $|a_j|\|W_{2,:,j}\|_2$, where a_j is its GELU activation and $W_{2,:,j}$ its output-projection column. We rank units by their mean contributions and report the smallest fraction accounting for 90% of the total. This estimate ignores cancellation between unit outputs and does not establish that the remaining units can be pruned without loss.

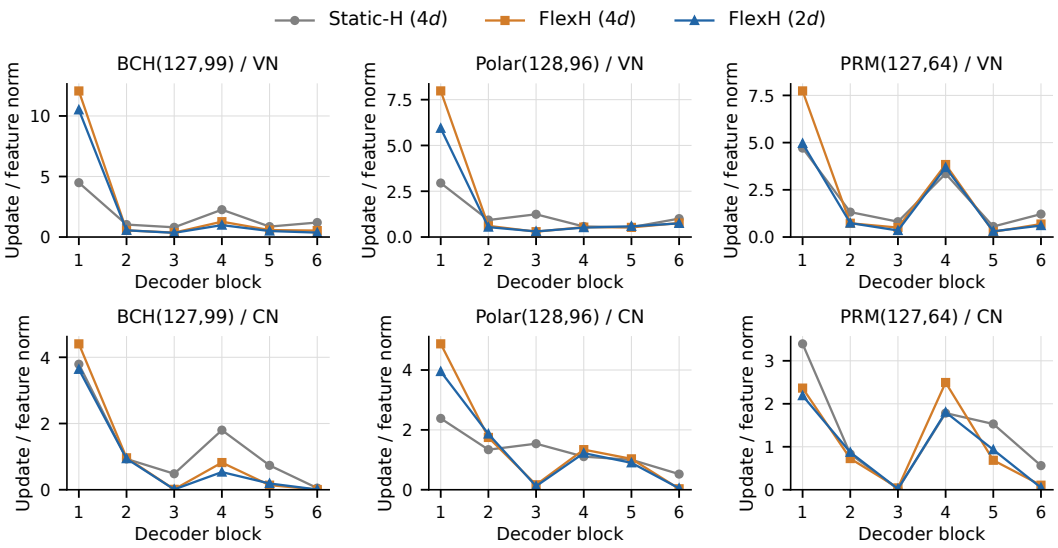


Figure 12: FFN update norms relative to pre-addition feature norms across all six EfficientMPT blocks for VN (top) and CN (bottom) tokens, averaged over 4–6 dB. Block 2 is the focus of the main-text analysis.

Across all six blocks, Figure 12 shows that FlexH redistributes relative FFN updates rather than reducing them everywhere. The first-block VN ratios increase; in Polar's second block, the CN ratio also rises from about 1.34 for Static-H $4d$ to 1.75 for FlexH $4d$ and 1.86 for FlexH $2d$. Thus, the smaller relative VN updates in the main text are part of a broader change across layers and paths. This supports a change in how FFNs are used, not a uniform reduction in their role throughout the decoder.

G Exploratory LDPC Extension

To examine whether the structural idea of FlexH can also support compact FFNs on sparse codes, Table 10 compares EfficientMPT with FlexH–EfficientMPT on LDPC(64,32), LDPC(121,70), and LDPC(512,256). We follow the same core principle as FlexH: channel reliabilities condition a dynamic, code-equivalent parity-check representation for each received word. For an LDPC code, however, forcing a complete identity-column basis would substantially densify the parity-check matrix and undermine its sparse graph structure. We therefore apply reliability-guided local row updates to the native sparse matrix, allowing its representation to change dynamically without requiring a full identity-column basis. FlexH–EfficientMPT then reduces the FFN hidden width from $4d$ to $2d$.

Despite using only half the FFN hidden width, FlexH–EfficientMPT retains BER and BLER close to EfficientMPT across 4–6 dB. Some points show a modest loss, while it gives slightly lower BER on LDPC(121,70) at 5 and 6 dB and improves both metrics on LDPC(512,256) at 6 dB. For each code, the parameter count falls from 1,092,993 to 698,241, or about 64% of the baseline, and the narrower FFN also reduces FLOPs. The LDPC extension therefore exhibits the same pattern as the long-Polar results: structural adaptation retains baseline-level performance, with occasional improvements, while using less learned and computational capacity. This consistency on sparse codes further supports our interpretation of FlexH as providing conditional structural capacity.

Table 10: LDPC extension at 4–6 dB, comparing EfficientMPT with FlexH–EfficientMPT. Each entry is BER / BLER. The better BER and BLER for each code and SNR are bold.

Code	Decoder	4 dB	5 dB	6 dB
LDPC(64,32)	EfficientMPT	$\mathbf{8.07\times10^{-4}}$ / $\mathbf{1.02\times10^{-2}}$	$\mathbf{3.29\times10^{-5}}$ / $\mathbf{5.48\times10^{-4}}$	$\mathbf{6.12\times10^{-7}}$ / $\mathbf{1.25\times10^{-5}}$
	FlexH–EfficientMPT	8.17×10^{-4} / 1.03×10^{-2}	4.50×10^{-5} / 6.40×10^{-4}	7.94×10^{-7} / 1.49×10^{-5}
LDPC(121,70)	EfficientMPT	$\mathbf{8.91\times10^{-4}}$ / $\mathbf{1.51\times10^{-2}}$	1.56×10^{-5} / $\mathbf{3.10\times10^{-4}}$	4.37×10^{-8} / $\mathbf{1.86\times10^{-6}}$
	FlexH–EfficientMPT	1.04×10^{-3} / 1.75×10^{-2}	$\mathbf{1.53\times10^{-5}}$ / 3.27×10^{-4}	$\mathbf{4.34\times10^{-8}}$ / 2.32×10^{-6}
LDPC(512,256)	EfficientMPT	$\mathbf{8.96\times10^{-7}}$ / $\mathbf{1.49\times10^{-4}}$	$\mathbf{1.09\times10^{-8}}$ / $\mathbf{5.59\times10^{-6}}$	1.58×10^{-8} / 8.08×10^{-6}
	FlexH–EfficientMPT	9.09×10^{-7} / 1.67×10^{-4}	1.13×10^{-8} / 5.77×10^{-6}	$\mathbf{1.35\times10^{-8}}$ / $\mathbf{6.92\times10^{-6}}$

H Reproducibility Details

H.1 Channel, Codes, and Training

All neural experiments use BPSK transmission over an AWGN channel. Training uses all-zero codewords, and BER/BLER evaluation samples general codewords. The Transformer training SNR set is $E_b/N_0 \in \{2, 3, 4, 5, 6, 7\}$ dB, and the reported test points span 1–6 dB. The residual-stream diagnostics follow the protocol in Appendix F.1.

The matched base/FlexH comparisons on BCH, Polar, and PRM use a common systematic base parity-check matrix for each code, obtained by binary row operations without column permutations. The exploratory LDPC experiments retain their native sparse base matrices.

Unless a table states otherwise, ECCT, CrossMPT, and EfficientMPT use six decoder layers, $d = 128$, and eight attention heads. The unprefixed base-decoder names use a $4d$ FFN, and the corresponding FlexH variants use a $2d$ FFN. Models are trained from scratch for 1000 epochs with batch size 128, Adam at an initial learning rate of 10^{-4}, and cosine annealing to 10^{-6}. Each epoch contains 1000 optimizer steps. The main BCH, Polar, and PRM models use ordinary binary cross-entropy with no distillation or auxiliary target. Training uses seed 42, and the checkpoint with the lowest training loss is selected.

H.2 Unified Evaluation Protocol

The main Transformer comparisons use a common post-training evaluator. At each SNR, sampling continues until both the number of received words is greater than 10^5 and the number of block errors is greater than 100, subject to a maximum of 10^9 words. This stopping rule is applied to BER and BLER together; the two metrics therefore come from the same decoded words. The logical inference batch size is 2048, and microbatching, when required by memory, does not change the statistical stopping rule.

ABP uses at most 20 iterations and $\alpha = 0.1$. Chase-II enumerates flip patterns on the 5, 3, 2, and 4 least reliable positions for BCH(63,36), BCH(63,45), BCH(63,51), and BCH(127,99), respectively. RPA uses all projections and at most three iterations on PRM(63,42). Chase-II and RPA are evaluated at 1–6 dB with a maximum of 5×10^7 words. ABP uses the same maximum at 1–3 dB and at most 10^6 words at 4–6 dB. The external ML files are taken from the channel-code database (Helmling et al., 2025); their reported FER is treated as BLER, and no BER is reconstructed from it.

H.3 Efficiency Measurements

Parameter counts come from the corresponding model definitions and match the saved checkpoints. FLOPs are measured per codeword at batch size one with the same profiler for each base/FlexH pair; the FlexH counts include its preprocessing. Peak allocated inference memory is measured on one NVIDIA GeForce RTX 5090 at the common logical batch size of 2048.