CR-CCM: COSET-REDUCED CONCATENATED-CODE MODELING FOR NEURAL DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformer-based neural decoders have achieved strong empirical performance under a unified formulation: binary linear codes are represented by parity-check matrices, and decoding is treated as bit-level noise prediction. This unified formulation is convenient, but it treats different code families in much the same way and can hide their structural advantages. For concatenated codes, the full-binary representation likewise obscures the distinct roles of the inner and outer codes. We propose Coset-Reduced Concatenated-Code Modeling (CR-CCM), a neural decoding framework tailored to algebraic concatenated codes. Building on classical inner-to-outer processing, CR-CCM uses a coset-coordinate decomposition to reformulate full-binary neural decoding as residual prediction on the binary-expanded outer graph. A unified soft interface connects different local estimators to syndrome-based neural decoders through tentative outer symbols and bitwise reliability. The neural decoder predicts the remaining errors before inner re-encoding. Relative to EfficientMPT, CR-CCM-EfficientMPT generally maintains or improves BER while substantially reducing computational cost. Average and maximum reductions are 75.8% and 94.9% in FLOPs, and 42.2% and 73.8% in peak training memory. At $\mathrm{SNR} = 6$ dB, BER is reduced by $1.83\times$ on average and up to approximately $106\times$.

1 INTRODUCTION

Error-correcting codes enable reliable communication over noisy channels. However, achieving near-maximum-likelihood performance for many code families still requires high computational cost. Early neural decoders followed two main directions. Some unfold iterative algorithms into trainable networks (Nachmani et al., 2016; Lugosch & Gross, 2017; Nachmani et al., 2018; Nachmani & Wolf, 2019; Buchberger et al., 2021). Others learn the decoding map with generic architectures (Cammerer et al., 2017; Gruber et al., 2017; Kim et al., 2018). Syndrome-based methods later shifted the task to channel-noise prediction. They use reliability and parity-check information, which helps the model generalize beyond memorized codewords (Bennatan et al., 2018). Transformer-based decoders build on this idea with parity-check-guided attention, cross-attention, and efficient message passing (Choukroun & Wolf, 2022; 2024; Park et al., 2025; 2024).

Despite strong empirical results, most Transformer-based decoders are designed as general decoders for binary linear codes. They represent a code mainly by its parity-check matrix (PCM) and use essentially the same parity-guided architecture across code families. As a result, cyclic, product, and concatenated codes all appear as ordinary binary graphs. This common representation is convenient, but it can hide the structural advantages of each code family.

Code families often contain useful regularities that a generic PCM does not make explicit. Prior work exploits shift equivalence and circulant relations in cyclic codes (Chen & Ye, 2021; Xiao et al., 2025; 2026). Other studies use section-wise revolving structure in quasi-cyclic LDPC codes (Zhang et al., 2024) and row–column dependencies in product codes (Li et al., 2026). These results show that exposing code-specific structure can make the neural prediction task better match the underlying construction.

Concatenated codes have long been important in communication systems (Forney, 1965; Ling & Xing, 2004). They combine an extension-field outer code with a binary inner code to build long, reliable codes without requiring a single large decoder. This gives a practical balance between

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.

reliability and decoding complexity. Classical decoding follows the same hierarchy. Each inner block is processed independently to produce tentative outer symbols, possibly with reliability information. The outer decoder then corrects the symbol sequence (Forney, 1966). Neural decoders have also been developed for concatenated systems, including multi-label component decoders (Leung et al., 2020) and soft inner estimation for deletion channels (Kargı & Duman, 2024). Transformer-based inner and outer decoders have been combined to handle synchronization errors (Streit et al., 2025). This established inner-to-outer processing order motivates our treatment of the neural task.

A direct application of a general binary Transformer decoder uses the full-binary PCM of the concatenated code. This preserves all constraints, but combines repeated local inner-code constraints and global outer parity checks in one enlarged graph. The inner constraints can be handled independently within each block. Expressing the remaining corrections in outer-symbol coordinates therefore offers a smaller task for neural prediction.

We therefore propose Coset-Reduced Concatenated-Code Modeling (CR-CCM), a neural decoding framework for algebraic concatenated codes. Using a coset-coordinate decomposition, CR-CCM reformulates full-binary neural decoding as residual correction on the binary-expanded outer graph. A unified soft interface produces tentative outer symbols and bitwise reliability. The neural decoder uses this reliability and the outer syndrome to predict the remaining errors. The corrected symbols are then re-encoded by the inner code. This interface supports different local soft-estimation methods and syndrome-based neural decoders. We adapt EfficientMPT (Park et al., 2026) to this task and call the resulting model CR-CCM-EfficientMPT.

The main contributions of this work are summarized as follows:

- We derive a coset-coordinate reduction that expresses the full weighted decoding problem in outer-symbol coordinates. With complete blockwise channel costs, this reduction preserves the feasible corrections and objective while removing repeated inner-code constraints from the neural graph.
- We connect local soft estimation, neural inputs, residual targets, and reconstruction through a common set of outer-bit coordinates. The resulting interface supports standard max-log, list-based, trellis-based, or posterior-based soft estimation and different syndrome-based neural decoders.
- We evaluate max-log, Chase-style, and trellis-based local interfaces and instantiate CR-CCM with EfficientMPT (Park et al., 2026), NBP, and HyperBP. Relative to EfficientMPT, CR-CCM-EfficientMPT generally maintains or improves BER, with average and maximum FLOPs reductions of 75.8% and 94.9%, and peak training memory reductions of 42.2% and 73.8%. At $\mathrm{SNR} = 6$ dB, the geometric-mean BER reduction is $1.83\times$, with a maximum of approximately $106\times$.

2 Background

2.1 Syndrome-Based Neural Noise Prediction

Let $\mathcal{C}$ be an $[n, k]$ binary linear code with generator matrix $\mathbf{G}$ and PCM $\mathbf{H}$ satisfying $\mathbf{G}\mathbf{H}^\top = \mathbf{0}$. A codeword $\mathbf{x} = \mathbf{m}\mathbf{G}$ is mapped by BPSK to $\mathbf{x}_s = \mathbf{1} - 2\mathbf{x}$ and received over AWGN as $\mathbf{y} = \mathbf{x}_s + \mathbf{z}$. Let $\mathbf{y}_b = \mathrm{bin}(\mathrm{sign}(\mathbf{y}))$, where $\mathrm{bin}(+1) = 0$ and $\mathrm{bin}(-1) = 1$, and define $\mathbf{e} = \mathbf{y}_b + \mathbf{x}$. Its observed syndrome is $\mathbf{s}(\mathbf{y}) = \mathbf{y}_b\mathbf{H}^\top = \mathbf{e}\mathbf{H}^\top$. Errors sharing a syndrome form an affine coset, so syndrome-based decoding selects a channel-compatible error from that coset and reconstructs $\widehat{\mathbf{x}} = \mathbf{y}_b + \widehat{\mathbf{e}}$.

Transformer-based decoders commonly represent the received word by its coordinate-wise magnitude $|\mathbf{y}|$ and syndrome $\mathbf{s}(\mathbf{y})$, and use the PCM to guide interactions between them (Choukroun & Wolf, 2022; 2024; Park et al., 2025; 2024). In the equivalent sign-domain noise formulation of syndrome-based neural decoding (Bennatan et al., 2018), $\mathbf{y} = \mathbf{x}_s \odot \widetilde{\mathbf{z}}_s$, where $\widetilde{\mathbf{z}}_s = \mathbf{y} \odot \mathbf{x}_s$. The decoder predicts $\widehat{\mathbf{z}}_s = f_\theta(|\mathbf{y}|, \mathbf{s}(\mathbf{y}); \mathbf{H})$ and reconstructs $\widehat{\mathbf{x}} = \mathrm{bin}(\mathrm{sign}(\mathbf{y} \odot \widehat{\mathbf{z}}_s))$. Reconstruction is correct when the predicted and true multiplicative noises have the same sign coordinate-wise. Since $\mathrm{sign}(\widetilde{z}_{s,j}) = -1$ if and only if $e_j = 1$, multiplicative-noise prediction and additive binary-error prediction describe the same hard-decision correction. We use the additive error notation below.

2.2 Concatenated Codes

We use *concatenated code* in its classical algebraic sense; broader serial or parallel coding architectures are covered only when they admit the symbol-wise outer–inner decomposition below.

We consider an algebraic concatenated code with an $[n_{\text{out}}, k_{\text{out}}]$ outer code over $\mathbb{F}_{2^{k_{\text{in}}}}$ and an $[n_{\text{in}}, k_{\text{in}}]$ binary inner code (Forney, 1965; Ling & Xing, 2004). The outer code has PCM $\mathbf{H}_{\text{out}} \in \mathbb{F}_{2^{k_{\text{in}}}}^{(n_{\text{out}}-k_{\text{out}})\times n_{\text{out}}}$. Fix an ordered binary basis of $\mathbb{F}_{2^{k_{\text{in}}}}$ and let $\beta : \mathbb{F}_{2^{k_{\text{in}}}} \to \mathbb{F}_2^{k_{\text{in}}}$ denote the corresponding coordinate map. Its componentwise extension is denoted by $\beta_{n_{\text{out}}}$. For an outer codeword $\overline{\mathbf{u}} \in \mathbb{F}_{2^{k_{\text{in}}}}^{n_{\text{out}}}$, the binary expansion is $\mathbf{u} = \beta_{n_{\text{out}}}(\overline{\mathbf{u}}) = (\mathbf{u}_1, \ldots, \mathbf{u}_{n_{\text{out}}})$, where $\mathbf{u}_i \in \mathbb{F}_2^{k_{\text{in}}}$. Each expanded symbol is encoded as the inner block $\mathbf{x}_i = \mathbf{u}_i \mathbf{G}_{\text{in}}$, where $\mathbf{G}_{\text{in}}$ and $\mathbf{H}_{\text{in}}$ are the inner generator and parity-check matrices. Stacking the n_{out} blocks gives a binary codeword of length $n_{\text{out}} n_{\text{in}}$ and dimension $k_{\text{out}} k_{\text{in}}$.

2.3 Binary Expansion of Concatenated Codes

The full-binary PCM of the concatenated code can be written as

$$\mathbf{H}_{\text{cc}} = \begin{bmatrix} \mathbf{I}_{n_{\text{out}}} \otimes \mathbf{H}_{\text{in}} \\ \mathbf{H}_{\text{out,bin}} (\mathbf{I}_{n_{\text{out}}} \otimes \mathbf{A})^{\top} \end{bmatrix}. \tag{1}$$

Here, $\mathbf{H}_{\text{out,bin}} \in \mathbb{F}_2^{(n_{\text{out}}-k_{\text{out}})k_{\text{in}} \times n_{\text{out}} k_{\text{in}}}$ is the binary matrix representing the linear map induced by $\mathbf{H}_{\text{out}}$ under the selected basis. The matrix $\mathbf{A} \in \mathbb{F}_2^{n_{\text{in}} \times k_{\text{in}}}$ is a right inverse of $\mathbf{G}_{\text{in}}$, satisfying $\mathbf{G}_{\text{in}} \mathbf{A} = \mathbf{I}_{k_{\text{in}}}$, and therefore $\mathbf{x}_i \mathbf{A} = \mathbf{u}_i$ for every inner codeword $\mathbf{x}_i$. The basis and coordinate order used by $\mathbf{A}$ and $\mathbf{H}_{\text{out,bin}}$ are assumed to match. In Equation (1), $\mathbf{I}_{n_{\text{out}}} \otimes \mathbf{H}_{\text{in}}$ enforces the inner-code checks independently on all blocks, while $\mathbf{H}_{\text{out,bin}} (\mathbf{I}_{n_{\text{out}}} \otimes \mathbf{A})^{\top}$ extracts their outer-bit coordinates and enforces the outer parity checks. The null space of $\mathbf{H}_{\text{cc}}$ coincides with the binary concatenated code, as shown in Lemma 1. This representation permits a standard binary neural decoder to process the concatenated code as one linear code.

3 Coset-Reduced Concatenated-Code Modeling

Figure 1 summarizes the three-step inference process. First, a local soft interface converts each received inner block into a tentative outer-symbol estimate together with a bitwise reliability vector. This estimate is not final; the neural decoder can still correct it. Next, the neural decoder uses the outer parity checks to predict the errors remaining in the tentative outer word. Finally, the corrected symbols are re-encoded by the inner code to produce the binary estimate. The local interface therefore handles the inner constraints, while the learned model runs only on the reduced outer graph.

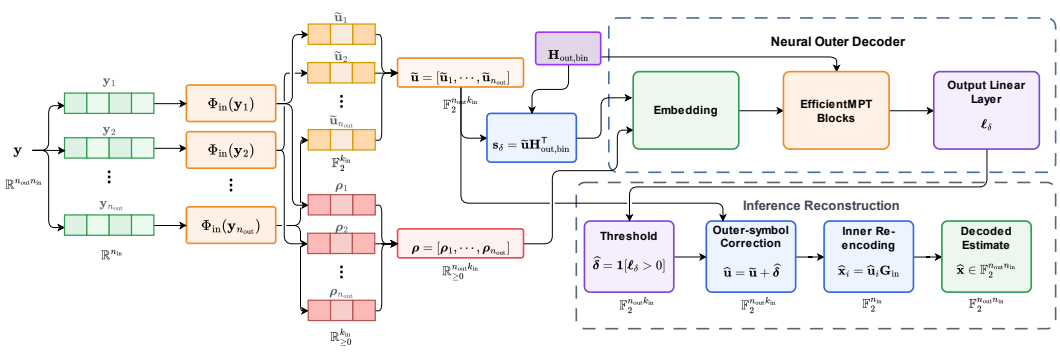


Figure 1: Inference-time overview of CR-CCM. Each received inner block is converted into a tentative outer symbol and bitwise reliability. A neural decoder then predicts the remaining errors before deterministic inner re-encoding.

3.1 LOCAL PROCESSING OF EACH INNER BLOCK

For the i-th inner-code block of the channel output $\mathbf{y}$, write $\mathbf{y}_i \in \mathbb{R}^{n_{\text{in}}}$. Its binary hard decision is $\mathbf{y}_{b,i} = \text{bin}(\text{sign}(\mathbf{y}_i))$. The corresponding inner syndrome is $\mathbf{s}_i = \mathbf{y}_{b,i}\mathbf{H}_{\text{in}}^\top$. A correction $\mathbf{e}_i$ must therefore satisfy $\mathbf{e}_i\mathbf{H}_{\text{in}}^\top = \mathbf{s}_i$. After this correction, $\mathbf{y}_{b,i} + \mathbf{e}_i$ is a valid inner codeword.

The key observation is that fixing $\mathbf{s}_i$ leaves only k_{in} binary coordinates to determine a compatible correction. The following proposition states this reduction formally.

Proposition 1. *Let $\mathcal{C}_{\text{in}}$ be the $[n_{\text{in}}, k_{\text{in}}]$ inner code defined by the full-row-rank matrices $\mathbf{G}_{\text{in}}$ and $\mathbf{H}_{\text{in}}$, where $\mathbf{G}_{\text{in}}\mathbf{H}_{\text{in}}^\top = \mathbf{0}$ and $\mathbf{G}_{\text{in}}\mathbf{A} = \mathbf{I}_{k_{\text{in}}}$. For any $\mathbf{v} \in \mathbb{F}_2^{n_{\text{in}}}$, let $\mathbf{s} = \mathbf{v}\mathbf{H}_{\text{in}}^\top$. There exists $\mathbf{B} \in \mathbb{F}_2^{(n_{\text{in}}-k_{\text{in}})\times n_{\text{in}}}$ satisfying $\mathbf{B}\mathbf{A} = \mathbf{0}$, $\mathbf{B}\mathbf{H}_{\text{in}}^\top = \mathbf{I}_{n_{\text{in}}-k_{\text{in}}}$, and $\mathbf{A}\mathbf{G}_{\text{in}} + \mathbf{H}_{\text{in}}^\top\mathbf{B} = \mathbf{I}_{n_{\text{in}}}$. Moreover, $\mathbf{v}$ has the unique decomposition*

$$\mathbf{v} = (\mathbf{v}\mathbf{A})\mathbf{G}_{\text{in}} + (\mathbf{v}\mathbf{H}_{\text{in}}^\top)\mathbf{B} = (\mathbf{v}\mathbf{A})\mathbf{G}_{\text{in}} + \mathbf{s}\mathbf{B}, \qquad \mathbf{v}\mathbf{H}_{\text{in}}^\top = \mathbf{s}. \tag{2}$$

The proof and an explicit construction of $\mathbf{B}$ are given in Appendix A.2. For fixed $\mathbf{s}_i$, the proposition shows that every candidate correction can be indexed by the length-k_{in} vector $\mathbf{a} = \mathbf{e}_i\mathbf{A}$. Let $\mathcal{E}_i = \{\mathbf{e} : \mathbf{e}\mathbf{H}_{\text{in}}^\top = \mathbf{s}_i\}$. Each $\mathbf{a} \in \mathbb{F}_2^{k_{\text{in}}}$ gives $\mathbf{e}_i(\mathbf{a}) := \mathbf{a}\mathbf{G}_{\text{in}} + \mathbf{s}_i\mathbf{B} \in \mathcal{E}_i$. This mapping is one-to-one, as proved in Corollary 1.

The hard decision produces the raw outer-symbol estimate $\mathbf{U}_i = \mathbf{y}_{b,i}\mathbf{A}$. For a short inner code with small k_{in}, one possible local implementation is a max-log rule. It selects the candidate with the lowest weighted flip cost. Under BPSK transmission over AWGN, this cost is

$$C_i(\mathbf{a}) = \sum_{j=1}^{n_{\text{in}}} |y_{i,j}| \big[\mathbf{e}_i(\mathbf{a})\big]_j, \qquad \mathbf{a}_i^{(0)} = \arg\min_{\mathbf{a}\in\mathbb{F}_2^{k_{\text{in}}}} C_i(\mathbf{a}), \qquad \widetilde{\mathbf{u}}_i = \mathbf{U}_i + \mathbf{a}_i^{(0)}. \tag{3}$$

The selected change gives the tentative symbol $\widetilde{\mathbf{u}}_i = \mathbf{U}_i + \mathbf{a}_i^{(0)}$. The weight $|y_{i,j}|$ makes reliable decisions more costly to flip than samples near zero.

We now connect this local representation to the original full concatenated-code decoding problem. Stack the hard decisions as $\mathbf{y}_b = (\mathbf{y}_{b,1}, \ldots, \mathbf{y}_{b,n_{\text{out}}})$. Write the block corrections as $\mathbf{e} = (\mathbf{e}_1, \ldots, \mathbf{e}_{n_{\text{out}}})$ and their labels as $\mathbf{a} = (\mathbf{a}_1, \ldots, \mathbf{a}_{n_{\text{out}}})$. The full weighted cost is $\mathcal{J}(\mathbf{e}) = \sum_{i=1}^{n_{\text{out}}} \sum_{j=1}^{n_{\text{in}}} |y_{i,j}| e_{i,j}$.

Stack the raw estimates and local minimizers as $\mathbf{U} = (\mathbf{U}_1, \ldots, \mathbf{U}_{n_{\text{out}}})$ and $\mathbf{a}^{(0)} = (\mathbf{a}_1^{(0)}, \ldots, \mathbf{a}_{n_{\text{out}}}^{(0)})$. The outer-check result after local selection is $\mathbf{s}_\delta = (\mathbf{U} + \mathbf{a}^{(0)})\mathbf{H}_{\text{out,bin}}^\top$. For $\mathbf{d} \in \mathbb{F}_2^{k_{\text{in}}}$, define the additional cost $D_i(\mathbf{d}) = C_i(\mathbf{a}_i^{(0)} + \mathbf{d}) - C_i(\mathbf{a}_i^{(0)})$.

Each inner block is now described by a k_{in}-bit label and a local cost. The next result shows that this replacement preserves both the feasible corrections and their channel costs.

Proposition 2. *Define the feasible sets $\Omega_e = \{\mathbf{e} : \mathbf{e}\mathbf{H}_{\text{cc}}^\top = \mathbf{y}_b\mathbf{H}_{\text{cc}}^\top\}$, $\Omega_a = \{\mathbf{a} : (\mathbf{U} + \mathbf{a})\mathbf{H}_{\text{out,bin}}^\top = \mathbf{0}\}$, and $\Omega_\delta = \{\boldsymbol{\delta} : \boldsymbol{\delta}\mathbf{H}_{\text{out,bin}}^\top = \mathbf{s}_\delta\}$. Then*

$$\min_{\mathbf{e}\in\Omega_e} \mathcal{J}(\mathbf{e}) = \min_{\mathbf{a}\in\Omega_a} \sum_{i=1}^{n_{\text{out}}} C_i(\mathbf{a}_i) = \sum_{i=1}^{n_{\text{out}}} C_i(\mathbf{a}_i^{(0)}) + \min_{\boldsymbol{\delta}\in\Omega_\delta} \sum_{i=1}^{n_{\text{out}}} D_i(\boldsymbol{\delta}_i). \tag{4}$$

The first equality replaces every length-n_{in} correction by its k_{in}-bit label $\mathbf{a}_i$. The fixed syndrome $\mathbf{s}_i$ already enforces the inner parity checks. The second equality writes $\mathbf{a}_i = \mathbf{a}_i^{(0)} + \boldsymbol{\delta}_i$. Thus, $\boldsymbol{\delta}_i = \mathbf{0}$ keeps the locally selected symbol, while a nonzero entry marks a bit that still needs correction. It remains to choose these changes so that the outer parity checks are satisfied.

The reduced objective in Proposition 2 is represented by the complete blockwise costs D_i after the inner-code variables and constraints have been eliminated. For the neural instantiation below, we derive a tentative outer symbol and a bitwise reliability vector from these costs as compact decoder inputs. The proof is given in Appendix A.4.

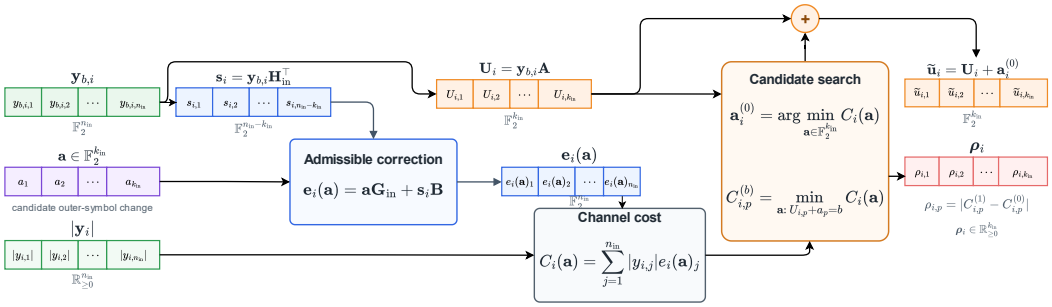


Figure 2: A max-log implementation for a low-dimensional short inner code. Candidate search uses the same costs for two operations: the upper formula selects the overall best candidate, giving $\widetilde{\mathbf{u}}_i$; the lower formula finds the minimum cost separately among candidates producing a 0 or a 1 at output position p. The absolute gap between these two costs gives $\rho_{i,p}$.

For output position p, group the candidates in Figure 2 by the corrected bit $U_{i,p} + a_p \in \{0, 1\}$. The superscript b in $C_{i,p}^{(b)}$ denotes this corrected bit value, not the value of a_p. All binary additions are modulo two. Taking the minimum cost within each group gives

$$C_{i,p}^{(0)} = \min_{\mathbf{a}:\, U_{i,p}+a_p=0} C_i(\mathbf{a}), \qquad C_{i,p}^{(1)} = \min_{\mathbf{a}:\, U_{i,p}+a_p=1} C_i(\mathbf{a}), \qquad \rho_{i,p} = \left| C_{i,p}^{(1)} - C_{i,p}^{(0)} \right|. \tag{5}$$

Each constrained minimum fixes only output bit p; all other candidate bits remain free. Thus, it returns a scalar cost for the best full-block explanation of that bit value. The candidate costs are reused, and no outer-code check is imposed at this local stage.

Because $\mathbf{a}_i^{(0)}$ is an overall minimizer, the group containing the selected bit $\widetilde{u}_{i,p}$ has minimum cost $C_i(\mathbf{a}_i^{(0)})$. Consequently, $\rho_{i,p}$ is the smallest extra cost required to reverse that selected bit, allowing other bits to change as well: $\rho_{i,p} = \min_{\mathbf{a}:\, a_p \neq a_{i,p}^{(0)}} C_i(\mathbf{a}) - C_i(\mathbf{a}_i^{(0)}) = \min_{\mathbf{d}:\, d_p=1} D_i(\mathbf{d})$. For example, if the selected bit is 1 with cost 0.4 and the best candidate producing 0 has cost 1.8, its reliability is 1.4. A small gap indicates an ambiguous bit; a large gap indicates strong local support. These bitwise summaries of D_i provide one reliability value for each variable node of the reduced outer graph. The likelihood derivation is given in Lemma 2. Together, Equations (3) and (5) define the local soft interface $\Phi_{\text{in}}(\mathbf{y}_i) = (\widetilde{\mathbf{u}}_i, \boldsymbol{\rho}_i)$, where $\widetilde{\mathbf{u}}_i \in \mathbb{F}_2^{k_{\text{in}}}$ and $\boldsymbol{\rho}_i \in \mathbb{R}_{\geq 0}^{k_{\text{in}}}$.

If $\mathbf{e}_i$ is the actual hard-decision error, then $\mathbf{a}_i^{\text{true}} = \mathbf{e}_i \mathbf{A}$ describes how that error changes the outer symbol. The tentative symbol differs from the transmitted outer symbol by $\boldsymbol{\delta}_i = \mathbf{a}_i^{\text{true}} + \mathbf{a}_i^{(0)}$. In plain terms, $\boldsymbol{\delta}_i$ marks exactly which bits of the tentative symbol were selected incorrectly by the local step. This is the quantity predicted by the neural decoder.

Code-family-specific soft estimators can also provide tentative symbols and bitwise reliability. Examples include Chase-style candidate estimation for block codes (Chase, 1972) and trellis-based soft-output computation (Bahl et al., 1974). Further examples are given in Appendix A.8.

3.2 Neural Correction on the Outer Code

Stacking the local outputs gives the tentative outer word $\widetilde{\mathbf{u}} = (\widetilde{\mathbf{u}}_1, \ldots, \widetilde{\mathbf{u}}_{n_{\text{out}}})$ and its bitwise reliability $\boldsymbol{\rho} = (\boldsymbol{\rho}_1, \ldots, \boldsymbol{\rho}_{n_{\text{out}}})$. Let $\boldsymbol{\delta} = \mathbf{u} + \widetilde{\mathbf{u}}$ denote the errors remaining in the tentative word. The vector $\mathbf{s}_\delta$ defined above is the outer-check result of $\widetilde{\mathbf{u}}$. Since the true outer word satisfies $\mathbf{u}\mathbf{H}_{\text{out,bin}}^\top = \mathbf{0}$, we obtain $\mathbf{s}_\delta = \boldsymbol{\delta}\mathbf{H}_{\text{out,bin}}^\top$.

The framework can be combined with different neural decoders. We use EfficientMPT as one implementation. This instantiation retains its backbone and training configuration but changes the variables, parity-check graph, and prediction target supplied to the model. The adapted decoder operates on the binary-expanded outer PCM $\mathbf{H}_{\text{out,bin}}$. Given $\boldsymbol{\rho}$ and $\mathbf{s}_\delta$, it outputs residual logits $\boldsymbol{\ell}_\delta = f_\theta(\boldsymbol{\rho}, \mathbf{s}_\delta; \mathbf{H}_{\text{out,bin}})$ and the estimate $\widehat{\boldsymbol{\delta}} = \mathbf{1}[\boldsymbol{\ell}_\delta > 0]$.

$\mathbf{s}_\delta$ shows which outer checks fail, while $\boldsymbol{\rho}$ highlights uncertain local choices. Together, they help the neural decoder locate corrections that are plausible and consistent with the outer code.

This also changes the prediction task. The full-binary formulation predicts errors on all transmitted bits, whereas CR-CCM predicts only the errors left in the tentative outer word. With $Q = n_{\text{out}}k_{\text{in}}$, the training target is $\boldsymbol{\delta}^{\text{true}} = \mathbf{u} + \tilde{\mathbf{u}}$. The loss is $\mathcal{L}_\delta = Q^{-1}\sum_{q=1}^{Q} \text{BCEWithLogits}(\ell_{\delta,q}, \delta_q^{\text{true}})$.

3.3 Reconstruction and Code Validity

The neural output specifies which tentative outer bits should be flipped, so the corrected outer word is $\widehat{\mathbf{u}} = \tilde{\mathbf{u}} + \widehat{\boldsymbol{\delta}}$.

The tentative symbols remain correctable. We apply the predicted residual before inner re-encoding to obtain the final binary estimate.

The following identity connects this re-encoding step to the correction variables above. Define $\widehat{\mathbf{a}}_i = \mathbf{a}_i^{(0)} + \widehat{\boldsymbol{\delta}}_i$. Then

$$\widehat{\mathbf{e}}_i = \widehat{\mathbf{a}}_i\mathbf{G}_{\text{in}} + \mathbf{s}_i\mathbf{B}, \qquad \widehat{\mathbf{x}}_i := \mathbf{y}_{b,i} + \widehat{\mathbf{e}}_i = (\mathbf{U}_i + \widehat{\mathbf{a}}_i)\mathbf{G}_{\text{in}} = \widehat{\mathbf{u}}_i\mathbf{G}_{\text{in}}.$$

The rightmost expression, $\widehat{\mathbf{x}}_i = \widehat{\mathbf{u}}_i\mathbf{G}_{\text{in}}$, is the actual reconstruction step. The preceding equalities show that it is equivalent to correcting the channel hard decision by $\widehat{\mathbf{e}}_i$. This guarantees that every reconstructed block is a valid inner codeword. Outer-code validity additionally requires $\widehat{\mathbf{u}}\mathbf{H}_{\text{out,bin}}^\top = \mathbf{0}$. Equivalently, the predicted flips must satisfy $\widehat{\boldsymbol{\delta}}\mathbf{H}_{\text{out,bin}}^\top = \mathbf{s}_\delta$. Under this condition, the complete output is a valid concatenated codeword. These statements are proved in Theorem 1.

3.4 Graph Reduction and Complexity

In the full-binary formulation of Equation (1), the neural decoder processes all n_{in} transmitted bits and all inner parity checks for every block, together with the outer parity checks. After local processing, CR-CCM gives the neural decoder only k_{in} tentative outer bits per block and the outer parity checks. The reduced graph therefore contains $Q = n_{\text{out}}k_{\text{in}}$ variable nodes and $R = (n_{\text{out}} - k_{\text{out}})k_{\text{in}}$ check nodes. Figure 3 shows the difference on a small concatenated code.

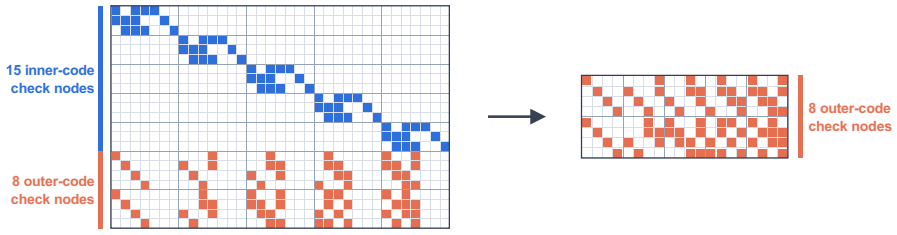


Figure 3: Full and reduced PCMs for $\text{RS}(5, 3; 4)$ concatenated with a $(7, 4)$ Hamming code. Local processing replaces 35 transmitted-bit variables and 15 inner checks by 20 outer-bit variables while retaining the 8 outer checks.

Counting both variable and check nodes gives

$$\begin{aligned} T_{\text{full}} &= n_{\text{out}}n_{\text{in}} + n_{\text{out}}(n_{\text{in}} - k_{\text{in}}) + (n_{\text{out}} - k_{\text{out}})k_{\text{in}}, \\ T_{\text{red}} &= n_{\text{out}}k_{\text{in}} + (n_{\text{out}} - k_{\text{out}})k_{\text{in}}, \qquad T_{\text{full}} - T_{\text{red}} \quad = 2n_{\text{out}}(n_{\text{in}} - k_{\text{in}}). \end{aligned} \tag{6}$$

The difference has two equal parts. Replacing each length-n_{in} received block by a length-k_{in} tentative symbol removes $n_{\text{out}}(n_{\text{in}} - k_{\text{in}})$ variable nodes, and removing the repeated inner parity checks saves the same number of check nodes.

For fixed network depth and embedding width, operations applied independently to each node scale linearly with the total number of nodes, whereas the dense variable–check message aggregation scales with the product of the variable- and check-node counts. When these two counts grow at a fixed ratio, this product is quadratic in the total number of nodes.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

The outer codes are Reed–Solomon (RS) and nonbinary LDPC (NB-LDPC) codes. The binary inner codes are BCH, LDPC, Golay, convolutional, extended Hamming, and Reed–Muller codes.

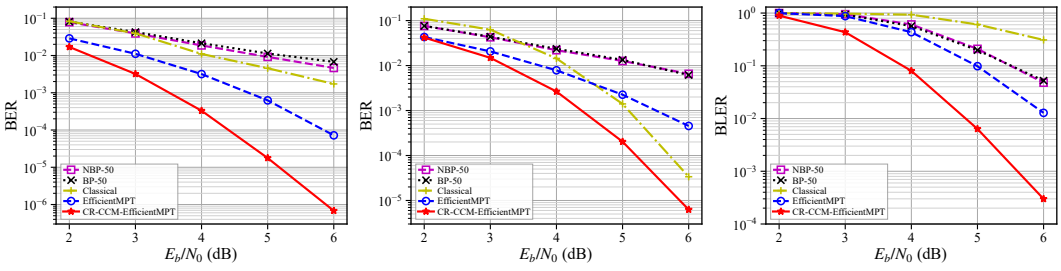
The neural decoders follow the training and evaluation protocol of EfficientMPT (Park et al., 2026). Throughout this paper, SNR denotes E_b/N_0. We train for 1000 epochs with 1000 minibatches of 128 samples per epoch using Adam. Training uses all-zero codewords over SNRs from 3 to 7 dB, whereas evaluation uses random codewords; the validity of this protocol follows from Proposition 3. Unless stated otherwise, the Transformer decoders use embedding dimension $d = 128$ and six decoding layers. The main experiments use the max-log interface in Section 3.1; alternative interfaces are evaluated in Appendix C. Further settings are given in Appendix B.

Our main comparison pairs EfficientMPT on the full-binary PCM with its CR-CCM instantiation, denoted CR-CCM-EfficientMPT. Classical decoding and NBP-50 provide additional references. The classical baseline uses family-matched inner decoders: Berlekamp–Massey (Massey, 1969) for BCH, Viterbi (Viterbi, 1967) for convolutional codes, majority logic or bounded-distance decoding for RM, and the corresponding bounded-distance decoder for the remaining inner codes. RS outer codes use reliability-aided errors-and-erasures decoding, while NB-LDPC outer codes use nonbinary sum-product decoding (Davey & MacKay, 1998). NBP-50 follows Nachmani et al. (2016) and runs for 50 iterations.

4.2 DECODING PERFORMANCE

Table 1: Decoding performance. Entries are $-\ln(\mathrm{BER})$ at the indicated E_b/N_0 in dB, and higher is better. The best result for each code and SNR is boldfaced.

Concatenated code	Classical			NBP-50			EfficientMPT			CR-CCM-EfficientMPT		
	4	5	6	4	5	6	4	5	6	4	5	6
$\mathrm{RS}(31,25;5)+\mathrm{BCH}(15,5)$	4.24	6.57	10.30	3.82	4.37	5.03	4.85	6.10	7.70	**5.93**	**8.50**	**11.98**
$\mathrm{RS}(63,55;6)+\mathrm{BCH}(14,6)$	3.60	6.11	**10.62**	4.47	5.81	7.68	**5.10**	**6.41**	8.06	**5.10**	**6.41**	8.07
$\mathrm{RS}(63,55;6)+\mathrm{LDPC}(12,6)$	2.66	3.03	3.46	4.20	4.96	5.99	**4.65**	**5.62**	6.75	**4.65**	**5.62**	**6.76**
$\mathrm{RS}(35,31;8)+\mathrm{CONV}(28,8)$	4.70	5.87	8.60	4.15	4.66	5.30	5.61	7.23	9.31	**5.83**	**7.56**	**9.69**
$\mathrm{RS}(107,99;8)+\mathrm{LDPC}(16,8)$	3.21	3.82	5.27	4.96	6.10	7.64	5.09	6.38	7.99	**5.26**	**6.56**	**8.16**
$\mathrm{RS}(63,55;6)+\mathrm{RM}(32,6)$	3.25	5.10	**10.37**	3.52	3.60	3.66	5.52	7.09	9.12	**5.79**	**7.45**	9.54
$\mathrm{NB\text{-}LDPC}(63,55;6)+\mathrm{BCH}(14,6)$	3.12	3.77	4.67	5.36	6.94	8.64	6.09	8.07	10.41	**7.02**	**9.69**	**12.90**
$\mathrm{NB\text{-}LDPC}(63,55;6)+\mathrm{LDPC}(12,6)$	4.61	5.55	6.70	5.51	7.09	8.75	6.24	8.45	11.06	**6.33**	**8.54**	**11.24**
$\mathrm{NB\text{-}LDPC}(63,55;6)+\mathrm{GOLAY}(12,6)$	4.19	5.09	6.27	5.75	7.34	8.81	6.77	9.25	12.13	**6.84**	**9.46**	**12.63**
$\mathrm{NB\text{-}LDPC}(46,38;7)+\mathrm{CONV}(14,7)$	5.08	5.43	6.24	5.77	7.12	8.28	6.79	9.16	11.99	**7.02**	**9.51**	**12.61**
$\mathrm{NB\text{-}LDPC}(107,99;8)+\mathrm{CONV}(16,8)$	6.50	7.51	7.71	5.10	6.51	7.84	6.87	9.43	12.59	**7.23**	**10.10**	**13.85**



(a) $\mathrm{NB\text{-}LDPC}(63,55;6)+\mathrm{BCH}(31,6)$ (b) $\mathrm{RS}(31,25;5)+\mathrm{BCH}(15,5)$ (c) $\mathrm{NB\text{-}LDPC}(63,55;6)+\mathrm{BCH}(14,6)$

Figure 4: BER and BLER comparisons on three concatenated codes. Panels (a) and (b) report BER, and panel (c) reports BLER; lower is better. BP-50 denotes conventional belief propagation with 50 iterations.

Table 1 shows that CR-CCM-EfficientMPT generally matches or improves EfficientMPT, with BER gains that vary across code combinations. At $\mathrm{SNR} = 6$ dB, the combined results in Tables 1, 4

and 5 give a $1.83\times$ geometric-mean BER reduction. For $\mathrm{RS}(31, 25; 5) + \mathrm{BCH}(15, 5)$, the gain in $-\ln(\mathrm{BER})$ increases from 1.08 at 4 dB to 4.28 at 6 dB. This corresponds to an approximately $72\times$ BER reduction at $\mathrm{SNR} = 6$ dB. For combinations with similar BER, Section 4.3 examines the corresponding computational savings.

The curves in Figure 4 show the same trend. At $\mathrm{SNR} = 6$ dB, EfficientMPT gives a BER of 7.20×10^{-5} on $\mathrm{NB\text{-}LDPC}(63, 55; 6) + \mathrm{BCH}(31, 6)$, whereas CR-CCM-EfficientMPT reaches 6.80×10^{-7}. This is an approximately $106\times$ reduction. Panel (c) also shows lower BLER than EfficientMPT for $\mathrm{NB\text{-}LDPC}(63, 55; 6) + \mathrm{BCH}(14, 6)$, so the improvement extends to complete-frame decoding.

The classical decoder in Table 1 provides a conventional performance reference. Additional neural-decoding results are given in Appendix D. Chase and trellis interfaces in Appendix C retain competitive decoding performance while reducing computational cost, including for larger inner-code dimensions. At $\mathrm{SNR} = 6$ dB, CR-CCM-NBP and CR-CCM-HyperBP reduce BER by median factors of 3.60 and 2.09, respectively, over their full-binary counterparts (Appendix F). Their efficiency results are given in Appendix G.

4.3 Complexity Analysis

Table 2: Efficiency comparison. F, M, and T denote FLOPs (G), peak training memory usage (MiB), and per-epoch training time (s), with batch sizes 1, 1, and 128. Parenthesized percentages are normalized to ECCT for F and M, and to EfficientMPT for T. Best results are boldfaced.

Concatenated code	ECCT		CrossMPT		EfficientMPT			CR-CCM-EfficientMPT		
	F	M	F	M	F	M	T	F	M	T
$\mathrm{RS}(63, 55; 6) + \mathrm{BCH}(14, 6)$	9.71 (100%)	1136 (100%)	6.38 (65.7%)	562 (49.5%)	4.72 (48.7%)	93 (8.2%)	189.3 (100%)	**0.98** (10.1%)	**51** (4.5%)	**49.6** (26.2%)
$\mathrm{RS}(63, 55; 6) + \mathrm{LDPC}(12, 6)$	7.08 (100%)	791 (100%)	4.77 (67.3%)	395 (49.9%)	3.63 (51.2%)	82 (10.4%)	150.5 (100%)	**0.98** (13.9%)	**51** (6.4%)	**50.8** (33.8%)
$\mathrm{RS}(35, 31; 8) + \mathrm{CONV}(28, 8)$	13.05 (100%)	1589 (100%)	8.45 (64.8%)	791 (49.8%)	6.09 (46.7%)	105 (6.6%)	174.5 (100%)	**0.71** (5.4%)	**46** (2.9%)	**50.1** (28.7%)
$\mathrm{RS}(107, 99; 8) + \mathrm{LDPC}(16, 8)$	27.50 (100%)	3658 (100%)	15.90 (57.8%)	1659 (45.4%)	10.94 (39.8%)	145 (4.0%)	290.3 (100%)	**2.17** (7.9%)	**71** (1.9%)	**52.5** (18.1%)
$\mathrm{RS}(63, 55; 6) + \mathrm{RM}(32, 6)$	50.86 (100%)	7118 (100%)	29.64 (58.3%)	3413 (47.9%)	19.32 (38.0%)	195 (2.7%)	485.5 (100%)	**0.98** (1.9%)	**51** (0.7%)	**49.8** (10.3%)
$\mathrm{NB\text{-}LDPC}(63, 55; 6) + \mathrm{BCH}(14, 6)$	9.71 (100%)	1136 (100%)	6.38 (65.7%)	562 (49.5%)	4.72 (48.7%)	93 (8.2%)	166.7 (100%)	**0.98** (10.1%)	**51** (4.5%)	**50.1** (30.1%)
$\mathrm{NB\text{-}LDPC}(63, 55; 6) + \mathrm{LDPC}(12, 6)$	7.08 (100%)	791 (100%)	4.77 (67.3%)	395 (49.9%)	3.63 (51.2%)	82 (10.4%)	102.3 (100%)	**0.98** (13.9%)	**51** (6.4%)	**47.9** (46.8%)
$\mathrm{NB\text{-}LDPC}(107, 99; 8) + \mathrm{CONV}(16, 8)$	27.50 (100%)	3658 (100%)	15.90 (57.8%)	1659 (45.4%)	10.94 (39.8%)	145 (4.0%)	261.4 (100%)	**2.17** (7.9%)	**71** (1.9%)	**53.8** (20.6%)

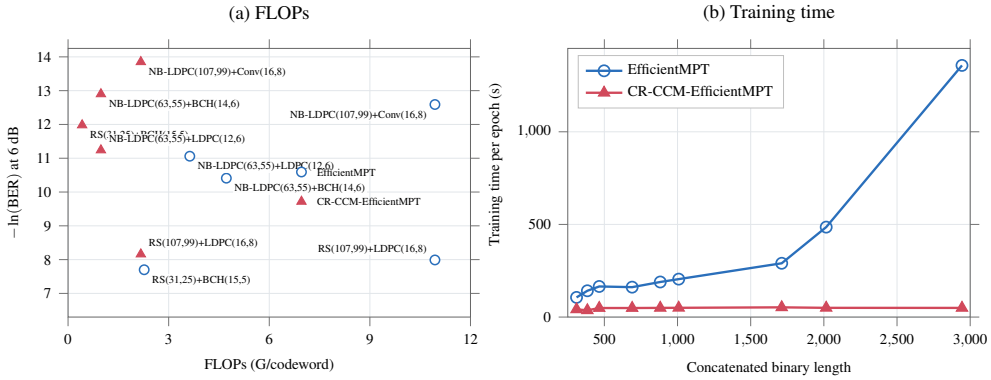


Figure 5: Efficiency comparison with EfficientMPT. (a) Decoding performance versus FLOPs at $\mathrm{SNR} = 6$ dB. (b) Per-epoch training time versus concatenated binary length.

Table 2 shows that CR-CCM-EfficientMPT consistently requires fewer FLOPs and less memory than EfficientMPT. On $\mathrm{RS}(63, 55; 6) + \mathrm{RM}(32, 6)$, FLOPs decrease from 19.32 to 0.98 G/codeword and memory usage from 195 to 51 MiB. These are the largest reductions, 94.9% and 73.8%, in the table.

For $\mathrm{RS}(63, 55; 6) + \mathrm{BCH}(14, 6)$, BER is nearly unchanged at $\mathrm{SNR} = 6$ dB (Table 1), yet FLOPs and memory usage fall by 79.2% and 45.2%.

Figure 5 compares decoding performance against FLOPs in panel (a) and per-epoch training time in panel (b). Across the plotted codes, training time varies much less for CR-CCM-EfficientMPT than for EfficientMPT. For $\mathrm{RS}(63, 55; 6) + \mathrm{RM}(32, 6)$, the per-epoch time decreases from 485.5 to 49.8 seconds, giving a $9.75\times$ speedup. Together, the panels show lower computational and training costs while maintaining or improving decoding performance.

These results agree with the graph reduction in Section 3.4. With the same $\mathrm{RS}(63, 55; 6)$ outer code, changing the inner code from $\mathrm{BCH}(14, 6)$ to $\mathrm{RM}(32, 6)$ increases EfficientMPT's FLOPs from 4.72 to 19.32 G/codeword. The corresponding CR-CCM-EfficientMPT costs are both approximately 0.98 G/codeword. The two reduced models use outer graphs of the same size, although the full-binary graphs differ substantially. Additional efficiency measurements are provided in Appendix E.

4.4 Ablation Study

Table 3: Component ablation at SNRs of 4, 5, and 6 dB. Best results are boldfaced.

Variant	$-\ln(\mathrm{BER})\uparrow$			BER $\downarrow$			BLER $\downarrow$		
	4	5	6	4	5	6	4	5	6
$\mathrm{RS}(31, 25; 5) + \mathrm{BCH}(15, 5)$									
EfficientMPT	4.85	6.10	7.70	7.84e-3	2.25e-3	4.54e-4	0.438	0.149	0.0319
Local only	4.84	6.09	7.69	7.88e-3	2.27e-3	4.56e-4	0.397	0.136	0.0290
CR-CCM-EfficientMPT w/o reliability	4.84	6.09	7.69	7.88e-3	2.27e-3	4.56e-4	0.397	0.136	0.0290
CR-CCM-EfficientMPT	**5.93**	**8.50**	**11.98**	**2.65e-3**	**2.03e-4**	**6.27e-6**	**0.0897**	**0.00795**	**2.79e-4**
$\mathrm{NB\text{-}LDPC}(107, 99; 8) + \mathrm{CONV}(16, 8)$									
EfficientMPT	6.87	9.43	12.59	1.04e-3	8.00e-5	3.41e-6	0.329	0.0399	0.00234
Local only	5.71	7.28	9.16	3.30e-3	6.90e-4	1.05e-4	0.665	0.213	0.0382
CR-CCM-EfficientMPT w/o reliability	5.83	7.49	9.51	2.95e-3	5.57e-4	7.43e-5	0.559	0.150	0.0227
CR-CCM-EfficientMPT	**7.23**	**10.10**	**13.85**	**7.21e-4**	**4.12e-5**	**9.70e-7**	**0.128**	**0.00847**	**2.08e-4**

Local only and the full model in Table 3 use the same local soft interface. Local only omits outer correction and re-encodes the tentative symbols. Their comparison measures the benefit of neural residual correction beyond local processing. The reliability-free variant retains the reduced graph and residual target. It is trained separately, with all reliability inputs fixed to one during both training and testing. Comparing it with Local only evaluates neural correction without reliability; comparing it with the full model isolates the contribution of bitwise reliability.

For RS–BCH, the reliability-free model performs almost identically to Local only, whereas the full model substantially improves BER and BLER. For NB-LDPC–convolutional codes, the syndrome and reduced graph alone provide a limited improvement; supplying bitwise reliability leads to a much larger gain. Thus, the local interface provides both an initial estimate and reliability information that helps the neural decoder correct the remaining errors. The EfficientMPT comparison evaluates the complete reformulation.

5 Conclusion

This paper proposes Coset-Reduced Concatenated-Code Modeling (CR-CCM) for neural decoding of algebraic concatenated codes. CR-CCM reformulates full-binary neural decoding as residual prediction on the binary-expanded outer graph. A coset-coordinate reduction provides the underlying weighted formulation with complete blockwise costs. A local soft interface supplies tentative outer symbols and bitwise reliability for this residual prediction task. A neural decoder uses the outer parity checks to predict the remaining errors before the corrected symbols are re-encoded. We instantiate the framework with EfficientMPT, NBP, and HyperBP. Relative to EfficientMPT, CR-CCM-EfficientMPT reduces FLOPs and peak training memory by 75.8% and 42.2% on average. At $\mathrm{SNR} = 6$ dB, BER is reduced by $1.83\times$ on geometric average and up to approximately $106\times$. These results show that exploiting concatenated-code structure can substantially lower neural decoding cost while generally maintaining or improving decoding performance.

AI USE STATEMENT

In preparing this manuscript, we used generative AI tools to assist with drafting and language polishing. We also used them as auxiliary reviewers to check the presentation and internal consistency of the proposed method and mathematical derivations.

REPRODUCIBILITY STATEMENT

The experimental setup, baseline implementations, and evaluation criteria are described in Section 4 and Appendix B. Proofs of the main mathematical claims are provided in Appendix A. The remaining appendices report additional interface, performance, and efficiency results. Code, configuration files, and trained models will be released upon publication.

REFERENCES

L. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate (corresp.). *IEEE Transactions on Information Theory*, 20(2):284–287, 1974. doi: 10.1109/TIT.1974.1055186.

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.

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

M.C. Davey and D. MacKay. Low-density parity check codes over gf(q). *IEEE Communications Letters*, 2(6):165–167, 1998. doi: 10.1109/4234.681360.

G Forney. Generalized minimum distance decoding. *IEEE Transactions on Information Theory*, 12 (2):125–131, 1966.

G David Forney. Concatenated codes. 1965.

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.

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.

San Ling and Chaoping Xing. *Coding Theory: A First Course*. Cambridge university press, 2004.

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.

J. Massey. Shift-register synthesis and bch decoding. *IEEE Transactions on Information Theory*, 15 (1):122–127, 1969. doi: 10.1109/TIT.1969.1054260.

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, Yongjune Kim, and Jong-Seon No. Crossmpt: Cross-attention message-passing transformer for error correcting codes, 2024. URL https://arxiv.org/abs/2405.01033.

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, 2025. doi: 10.1109/JSAC.2025.3559154.

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.

A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967. doi: 10.1109/TIT.1967.1054010.

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.

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 Additional Method Details

This section proves the binary representation, coset-coordinate reduction, and reconstruction used in the main text. It also explains how different local soft estimators fit the common interface and gives the complexity of the dense EfficientMPT implementation.

A.1 Binary Expansion and Validity of the Full PCM

This subsection verifies that the matrix $\mathbf{H}_{\mathrm{cc}}$ in Equation (1) correctly represents the concatenated code. Under the fixed basis from Section 2.2, let β_t denote componentwise binary expansion of a length-t vector. The binary matrix $\mathbf{H}_{\mathrm{out,bin}}$ then implements the same outer parity checks, so every $\overline{\mathbf{u}} \in \mathbb{F}_{2^{k_{\mathrm{in}}}}^{n_{\mathrm{out}}}$ satisfies

$$\beta_{n_{\mathrm{out}}-k_{\mathrm{out}}}(\overline{\mathbf{u}}\mathbf{H}_{\mathrm{out}}^{\top}) = \beta_{n_{\mathrm{out}}}(\overline{\mathbf{u}})\mathbf{H}_{\mathrm{out,bin}}^{\top}.$$

Lemma 1. *Assume that $\mathbf{G}_{\mathrm{in}}$ and $\mathbf{H}_{\mathrm{in}}$ have full row rank. For a stacked binary vector $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_{n_{\mathrm{out}}})$, the condition $\mathbf{x}\mathbf{H}_{\mathrm{cc}}^{\top} = \mathbf{0}$ holds if and only if every block is a valid inner codeword and the extracted binary outer word $\mathbf{u} = \mathbf{x}(\mathbf{I}_{n_{\mathrm{out}}} \otimes \mathbf{A})$ is the binary expansion of a valid outer codeword. Consequently, the null space of Equation* (1) *coincides with the binary concatenated code.*

Proof. The upper block of $\mathbf{H}_{\mathrm{cc}}$ gives $\mathbf{x}_i\mathbf{H}_{\mathrm{in}}^{\top} = \mathbf{0}$ for every i. Because $\mathbf{G}_{\mathrm{in}}$ has full row rank, this is equivalent to $\mathbf{x}_i = \mathbf{m}_i\mathbf{G}_{\mathrm{in}}$ for a unique $\mathbf{m}_i \in \mathbb{F}_2^{k_{\mathrm{in}}}$. Applying the right inverse gives $\mathbf{x}_i\mathbf{A} = \mathbf{m}_i$, so the blocks of $\mathbf{u}$ are precisely these inner messages. The lower block gives

$$\mathbf{x}(\mathbf{I}_{n_{\mathrm{out}}} \otimes \mathbf{A})\mathbf{H}_{\mathrm{out,bin}}^{\top} = \mathbf{u}\mathbf{H}_{\mathrm{out,bin}}^{\top} = \mathbf{0},$$

which is equivalent by the binary-expansion identity above to the extension-field outer parity checks. The converse follows by reversing these two implications. □

A.2 Proof of the Inner-Coordinate Decomposition

The goal is to construct the matrix $\mathbf{B}$ used in Equation (2). We show that the k_{in} outer-symbol coordinates and the $n_{\mathrm{in}} - k_{\mathrm{in}}$ parity-check outcomes form a complete coordinate system for any length-n_{in} binary vector.

Proof of Proposition 1. Let $\mathbf{M} = [\,\mathbf{A}\ \ \mathbf{H}_{\mathrm{in}}^{\top}\,] \in \mathbb{F}_2^{n_{\mathrm{in}} \times n_{\mathrm{in}}}$ and suppose that $\mathbf{M}[\mathbf{c}^{\top}\ \mathbf{d}^{\top}]^{\top} = \mathbf{0}$. Left multiplication by $\mathbf{G}_{\mathrm{in}}$ gives $\mathbf{c} = \mathbf{0}$ because $\mathbf{G}_{\mathrm{in}}\mathbf{A} = \mathbf{I}_{k_{\mathrm{in}}}$ and $\mathbf{G}_{\mathrm{in}}\mathbf{H}_{\mathrm{in}}^{\top} = \mathbf{0}$. It then follows that $\mathbf{H}_{\mathrm{in}}^{\top}\mathbf{d} = \mathbf{0}$, and the full row rank of $\mathbf{H}_{\mathrm{in}}$ gives $\mathbf{d} = \mathbf{0}$. Hence $\mathbf{M}$ is nonsingular.

Since $\mathbf{G}_{\mathrm{in}}\mathbf{M} = [\,\mathbf{I}_{k_{\mathrm{in}}}\ \mathbf{0}\,]$, the first k_{in} rows of $\mathbf{M}^{-1}$ are $\mathbf{G}_{\mathrm{in}}$. We may therefore partition the inverse as

$$\mathbf{M}^{-1} = \begin{bmatrix} \mathbf{G}_{\mathrm{in}} \\ \mathbf{B} \end{bmatrix}.$$

The lower block row of $\mathbf{M}^{-1}\mathbf{M} = \mathbf{I}$ gives $\mathbf{B}\mathbf{A} = \mathbf{0}$ and $\mathbf{B}\mathbf{H}_{\mathrm{in}}^{\top} = \mathbf{I}_{n_{\mathrm{in}}-k_{\mathrm{in}}}$, while $\mathbf{M}\mathbf{M}^{-1} = \mathbf{I}$ gives $\mathbf{A}\mathbf{G}_{\mathrm{in}} + \mathbf{H}_{\mathrm{in}}^{\top}\mathbf{B} = \mathbf{I}_{n_{\mathrm{in}}}$. Therefore, for every $\mathbf{v} \in \mathbb{F}_2^{n_{\mathrm{in}}}$,

$$\mathbf{v} = \mathbf{v}\mathbf{M}\mathbf{M}^{-1} = (\mathbf{v}\mathbf{A})\mathbf{G}_{\mathrm{in}} + (\mathbf{v}\mathbf{H}_{\mathrm{in}}^{\top})\mathbf{B}.$$

Applying $\mathbf{A}$ and $\mathbf{H}_{\mathrm{in}}^{\top}$ to this representation recovers the two coordinates $\mathbf{v}\mathbf{A}$ and $\mathbf{v}\mathbf{H}_{\mathrm{in}}^{\top}$, respectively, so the decomposition is unique. The matrix $\mathbf{B}$ is obtained once from this binary matrix inverse and reused for every received block. □

A.3 One-to-One Parameterization of Candidate Corrections

Once the inner parity-check outcome $\mathbf{s}$ is fixed, only k_{in} free coordinates remain. The following result establishes a one-to-one correspondence between these coordinates and the length-n_{in} correction patterns with that parity-check outcome.

Corollary 1. *For any $\mathbf{s} \in \mathbb{F}_2^{n_{\mathrm{in}}-k_{\mathrm{in}}}$, define $\mathcal{E}(\mathbf{s}) = \{\mathbf{e} \in \mathbb{F}_2^{n_{\mathrm{in}}} : \mathbf{e}\mathbf{H}_{\mathrm{in}}^{\top} = \mathbf{s}\}$. The map*

$$\psi_{\mathbf{s}} : \mathbb{F}_2^{k_{\mathrm{in}}} \longrightarrow \mathcal{E}(\mathbf{s}), \qquad \psi_{\mathbf{s}}(\mathbf{a}) = \mathbf{a}\mathbf{G}_{\mathrm{in}} + \mathbf{s}\mathbf{B},$$

is a bijection. Hence every $\mathbf{a} \in \mathbb{F}_2^{k_{\mathrm{in}}}$ uniquely identifies one correction compatible with $\mathbf{s}$, and every compatible correction has one such representation.

Proof. The identities in Proposition 1 give

$$\psi_{\mathbf{s}}(\mathbf{a})\mathbf{H}_{\text{in}}^{\top} = \mathbf{s}, \qquad \psi_{\mathbf{s}}(\mathbf{a})\mathbf{A} = \mathbf{a}.$$

Thus, the map takes values in $\mathcal{E}(\mathbf{s})$ and is injective. Conversely, every $\mathbf{e} \in \mathcal{E}(\mathbf{s})$ satisfies $\mathbf{e} = (\mathbf{e}\mathbf{A})\mathbf{G}_{\text{in}} + \mathbf{s}\mathbf{B}$ by Equation (2), proving surjectivity. □

A.4 Removing the Repeated Inner-Code Constraints

We now substitute the k_{in}-bit representation of each candidate correction into the full-binary decoding problem. The proof has two steps: first remove the already-satisfied inner constraints, and then measure every remaining change relative to the tentative symbol selected locally.

Proof of Proposition 2. The inner-code constraints require every feasible $\mathbf{e}_i$ to satisfy $\mathbf{e}_i\mathbf{H}_{\text{in}}^{\top} = \mathbf{s}_i$, where $\mathbf{s}_i = \mathbf{y}_{b,i}\mathbf{H}_{\text{in}}^{\top}$. By Corollary 1, each such error has the unique form $\mathbf{e}_i(\mathbf{a}_i) = \mathbf{a}_i\mathbf{G}_{\text{in}} + \mathbf{s}_i\mathbf{B}$, and its weighted contribution is $C_i(\mathbf{a}_i)$.

For the outer-check rows of $\mathbf{H}_{\text{cc}}$, the remaining constraint becomes

$$\mathbf{e}(\mathbf{I}_{n_{\text{out}}} \otimes \mathbf{A})\mathbf{H}_{\text{out,bin}}^{\top} = \mathbf{y}_b(\mathbf{I}_{n_{\text{out}}} \otimes \mathbf{A})\mathbf{H}_{\text{out,bin}}^{\top}.$$

Since $\mathbf{e}_i(\mathbf{a}_i)\mathbf{A} = \mathbf{a}_i$ and $\mathbf{y}_{b,i}\mathbf{A} = \mathbf{U}_i$, this condition is equivalent to $(\mathbf{U} + \mathbf{a})\mathbf{H}_{\text{out,bin}}^{\top} = \mathbf{0}$. The coset bijection is blockwise, so it preserves the complete feasible set and proves the first equality in Equation (4).

Finally, the change of variables $\boldsymbol{\delta} = \mathbf{a} + \mathbf{a}^{(0)}$ is bijective. Its constraint is

$$\boldsymbol{\delta}\mathbf{H}_{\text{out,bin}}^{\top} = (\mathbf{U} + \mathbf{a}^{(0)})\mathbf{H}_{\text{out,bin}}^{\top} = \mathbf{s}_{\delta},$$

and $C_i(\mathbf{a}_i^{(0)} + \boldsymbol{\delta}_i) = C_i(\mathbf{a}_i^{(0)}) + D_i(\boldsymbol{\delta}_i)$. Summing over blocks proves the second equality. □

The same change of variables works for any tentative symbol $\widetilde{\mathbf{u}}_i$: set $\mathbf{a}_i^{(0)} = \mathbf{U}_i + \widetilde{\mathbf{u}}_i$ and define D_i relative to this estimate. With the complete block costs retained, the feasible set and cost identity remain unchanged. Choosing a minimum-cost $\mathbf{a}_i^{(0)}$, as in the max-log realization, additionally ensures $D_i \geq 0$. Only in that case does D_i measure the extra cost above the local optimum. The residual coordinates and reconstruction therefore remain applicable when the local estimator is replaced.

A.5 Why the Cost Gap Measures Reliability

For the max-log realization, the gap between two constrained channel costs provides the bitwise reliability. The following lemma derives this gap from the AWGN likelihood.

Lemma 2. *Assume BPSK transmission over AWGN with variance σ^2. An admissible correction $\mathbf{e}_i(\mathbf{a}) \in \mathcal{E}_i$ defines the candidate inner codeword $\mathbf{x}_i(\mathbf{a}) = \mathbf{y}_{b,i} + \mathbf{e}_i(\mathbf{a})$. Its negative log-likelihood satisfies*

$$-\log p(\mathbf{y}_i \mid \mathbf{x}_i(\mathbf{a})) = \text{const} + \frac{2}{\sigma^2} C_i(\mathbf{a}). \tag{7}$$

Under the standard equal-prior convention, the max-log LLR of outer-bit coordinate p is

$$L_{i,p}^{\text{max-log}} = \frac{2}{\sigma^2}\left(C_{i,p}^{(1)} - C_{i,p}^{(0)}\right). \tag{8}$$

Thus, $\widetilde{u}_{i,p}$ selects the lower-cost hypothesis and $\rho_{i,p}$ is its unnormalized max-log reliability magnitude.

Proof. Let $r_{i,j} = 1 - 2y_{b,i,j} = \text{sign}(y_{i,j})$. The BPSK signal associated with the candidate error bit $e_{i,j}$ is $r_{i,j}(1 - 2e_{i,j})$. Therefore,

$$(y_{i,j} - r_{i,j}(1 - 2e_{i,j}))^2 = (|y_{i,j}| - 1)^2 + 4|y_{i,j}|e_{i,j}.$$

Substitution into the Gaussian negative log-likelihood proves Equation (7). Minimizing this likelihood separately under the two constraints $U_{i,p} + a_p = 0$ and $U_{i,p} + a_p = 1$ gives $C_{i,p}^{(0)}$ and $C_{i,p}^{(1)}$. Replacing each log-sum-exp by its largest term yields Equation (8). Because $\mathbf{a}_i^{(0)}$ is the selected minimum-cost candidate, each bit of $\widetilde{\mathbf{u}}_i$ belongs to a lower-cost constrained hypothesis; ties give zero reliability and may be resolved deterministically. □

For a short inner code, the max-log rule obtains these two bit-hypothesis costs directly. Appendix A.8 describes other soft-estimation procedures that produce the same output format.

A.6 Codeword Symmetry

Training uses all-zero codewords, whereas evaluation uses random codewords. The next proposition gives a sufficient symmetry condition for this protocol. The proof verifies it for the direct max-log interface and explains the corresponding conditions for Chase-style and trellis-based soft estimates.

Proposition 3. *Under BPSK transmission over AWGN, suppose the local soft interface is translation-equivariant: for each block, the joint distribution of $(\boldsymbol{\rho}_i, \mathbf{u}_i + \widetilde{\mathbf{u}}_i)$ depends only on the channel error and not on the transmitted outer symbol $\mathbf{u}_i$. Then the joint distribution of the decoder inputs $(\boldsymbol{\rho}, \mathbf{s}_\delta)$ and target $\boldsymbol{\delta}$ is independent of the transmitted concatenated codeword. The direct max-log interface satisfies this condition when ties are resolved by a fixed rule in correction coordinates.*

Proof. Under the stated condition, $\boldsymbol{\delta}_i = \mathbf{u}_i + \widetilde{\mathbf{u}}_i$ and $\boldsymbol{\rho}_i$ are codeword-independent, and so is $\mathbf{s}_\delta = \boldsymbol{\delta}\mathbf{H}_{\mathrm{out,bin}}^\top$. This proves the conclusion. We next verify the condition for the max-log interface.

Write the hard decision in block i as $\mathbf{y}_{b,i} = \mathbf{x}_i + \mathbf{e}_i$. Since $\mathbf{x}_i\mathbf{A} = \mathbf{u}_i$, we have

$$\mathbf{U}_i = \mathbf{y}_{b,i}\mathbf{A} = \mathbf{u}_i + \mathbf{e}_i\mathbf{A}, \qquad \mathbf{s}_i = \mathbf{e}_i\mathbf{H}_{\mathrm{in}}^\top.$$

For a binary-input symmetric channel, the joint law of $(|\mathbf{y}_i|, \mathbf{e}_i)$ is codeword-independent. Hence the admissible set $\mathcal{E}_i$ and the cost table $C_i(\mathbf{a})$ depend only on the channel-error realization. A fixed tie rule in correction coordinates makes the selected minimizer $\mathbf{a}_i^{(0)}$ codeword-independent as well. A change in $\mathbf{u}_i$ only permutes the labels of the two constrained hypotheses defining $C_{i,p}^{(0)}$ and $C_{i,p}^{(1)}$, so their absolute gap $\rho_{i,p}$ is unchanged. Finally,

$$\boldsymbol{\delta}_i = \mathbf{u}_i + \widetilde{\mathbf{u}}_i = \mathbf{e}_i\mathbf{A} + \mathbf{a}_i^{(0)}, \qquad \mathbf{s}_\delta = \boldsymbol{\delta}\mathbf{H}_{\mathrm{out,bin}}^\top,$$

which are also codeword-independent. □

For Chase-style estimation, the same argument applies when the candidate correction list and its ordering depend only on the hard-decision syndrome and channel magnitudes. Codeword translation then leaves the list and its costs unchanged. For a trellis that represents the inner code, translation permutes the complete paths. With the corresponding sign change of the observations, their likelihood metrics are preserved. The two bit-hypothesis costs are either unchanged or exchanged, so their absolute gap is preserved. The selected path transforms accordingly when ties are resolved consistently under translation. Under continuous AWGN, likelihood ties between distinct codewords have probability zero, so this path-selection condition holds almost surely.

A.7 Validity of the Reconstructed Codeword

Reconstruction inner-encodes every corrected outer symbol. Each reconstructed block is therefore a valid inner codeword. The theorem below shows that the full concatenated word is valid if and only if the predicted flips make the corrected outer word satisfy its parity checks.

Theorem 1. *For any predicted residual $\widehat{\boldsymbol{\delta}}$, let $\widehat{\mathbf{a}}_i = \mathbf{a}_i^{(0)} + \widehat{\boldsymbol{\delta}}_i$, $\widehat{\mathbf{e}}_i = \widehat{\mathbf{a}}_i\mathbf{G}_{\mathrm{in}} + \mathbf{s}_i\mathbf{B}$, and $\widehat{\mathbf{u}} = \widetilde{\mathbf{u}} + \widehat{\boldsymbol{\delta}}$. Then every reconstructed block satisfies*

$$\mathbf{y}_{b,i} + \widehat{\mathbf{e}}_i = \widehat{\mathbf{u}}_i\mathbf{G}_{\mathrm{in}} \in \mathcal{C}_{\mathrm{in}}.$$

The stacked reconstruction is a valid concatenated codeword if and only if

$$\widehat{\boldsymbol{\delta}}\mathbf{H}_{\mathrm{out,bin}}^\top = \mathbf{s}_\delta. \tag{9}$$

In particular, $\widehat{\boldsymbol{\delta}} = \boldsymbol{\delta}$ reconstructs the transmitted concatenated codeword.

Proof. Applying Equation (2) to the hard decision gives $\mathbf{y}_{b,i} = \mathbf{U}_i\mathbf{G}_{\mathrm{in}} + \mathbf{s}_i\mathbf{B}$. Since addition is over $\mathbb{F}_2$,

$$\mathbf{y}_{b,i} + \widehat{\mathbf{e}}_i = (\mathbf{U}_i + \widehat{\mathbf{a}}_i)\mathbf{G}_{\mathrm{in}} = (\widetilde{\mathbf{u}}_i + \widehat{\boldsymbol{\delta}}_i)\mathbf{G}_{\mathrm{in}} = \widehat{\mathbf{u}}_i\mathbf{G}_{\mathrm{in}},$$

which proves deterministic inner-code validity. By Lemma 1, global concatenated-code validity is then equivalent to $\widehat{\mathbf{u}}\mathbf{H}_{\mathrm{out,bin}}^\top = \mathbf{0}$. Using $\widehat{\mathbf{u}} = \widetilde{\mathbf{u}} + \widehat{\boldsymbol{\delta}}$ and $\mathbf{s}_\delta = \widetilde{\mathbf{u}}\mathbf{H}_{\mathrm{out,bin}}^\top$ gives Equation (9). Finally, $\boldsymbol{\delta} = \mathbf{u} + \widetilde{\mathbf{u}}$ implies $\widehat{\mathbf{u}} = \mathbf{u}$ whenever $\widehat{\boldsymbol{\delta}} = \boldsymbol{\delta}$. □

A.8 Alternative Local Soft Interfaces

The framework connects a local soft estimator to the neural decoder through tentative outer-symbol bits and their bitwise reliability. The max-log realization in the main text is one way to obtain these outputs. The following coordinate convention also allows posterior-based, Chase-style, and trellis-based estimates to use the same residual task and reconstruction.

Required coordinate convention. All local implementations must use the same outer-symbol basis and coordinate order as $\mathbf{H}_{\mathrm{out,bin}}$. The right inverse $\mathbf{A}$ fixes these coordinates: $\mathbf{u}_i = \mathbf{x}_i\mathbf{A}$, and a correction $\mathbf{e}_i$ changes them by $\mathbf{a}_i = \mathbf{e}_i\mathbf{A}$. In our experiments, $\mathbf{G}_{\mathrm{in}}$ contains selected identity columns and $\mathbf{A}$ selects the corresponding information coordinates, so soft outputs for these positions can be used directly. For a general right inverse, the estimator must evaluate the binary combinations that define $\mathbf{x}_i\mathbf{A}$. This requires probabilities or candidate costs for those combinations, rather than a linear transformation of individual inner-bit LLRs.

APP or LLR output. Suppose a local soft-estimation module provides the bitwise log-likelihood ratio. It can be converted to the common output representation as

$$L_{i,p} = \log \frac{\Pr(u_{i,p} = 0 \mid \mathbf{y}_i)}{\Pr(u_{i,p} = 1 \mid \mathbf{y}_i)}, \qquad \widetilde{u}_{i,p} = \mathbf{1}[L_{i,p} < 0], \qquad \rho_{i,p} = |L_{i,p}|.$$

The sign selects the preferred local bit, and the magnitude measures the strength of that preference. This gives a posterior-based alternative to the max-log rule in the main text.

Chase-style candidate soft estimation for BCH inner codes. A Chase-style procedure forms a limited list of low-order flip patterns from the least reliable positions of the inner hard decision and completes them so that they satisfy the required inner parity checks. The lowest-cost candidate supplies $\widetilde{\mathbf{u}}_i$, while the best costs under the two values of each outer bit supply $\boldsymbol{\rho}_i$. This computes the max-log quantities over a restricted candidate list.

Trellis-based soft-output estimation for structured inner codes. When the inner code has a manageable trellis, a soft-output recursion computes the preferred path and the best path under each value of an outer-bit coordinate. The preferred path supplies $\widetilde{\mathbf{u}}_i$, and the difference between the two constrained path metrics supplies $\rho_{i,p}$. This is the trellis counterpart of the two hypothesis costs in Equation (5).

Chase lists and trellises describe how the soft estimates are computed; either can be used with max-log metrics. These implementations share an output format, but need not produce identical estimates or reliability values. For a fixed code, replacing the local estimator leaves the outer graph and the definition of the residual task unchanged. Each neural instance is trained and evaluated with its chosen interface.

A.9 Complexity of the Dense Implementation

The current EfficientMPT implementation uses dense matrix operations to exchange messages between every variable node and check node. In the reduced graph, $Q = n_{\mathrm{out}}k_{\mathrm{in}}$ is the number of tentative outer-bit variables and $R = (n_{\mathrm{out}} - k_{\mathrm{out}})k_{\mathrm{in}}$ is the number of binary outer checks. For L layers and embedding width d, its leading neural cost is

$$C^{\mathrm{dense}}_{\mathrm{neural}} = \mathcal{O}\big(L\big[(Q+R)d^2 + QRd\big] + QRd\big)\,.$$

The first term contains operations applied independently to each node and therefore scales linearly with $Q + R$. The QRd contribution in each layer comes from the two directional dense aggregations up to a constant factor, and the final QRd term is the output-side check-to-variable projection. When Q and R grow at a fixed ratio, these aggregation terms are quadratic in the total node count. For EfficientMPT on the full-binary graph, the same expression uses $Q_{\mathrm{full}} = n_{\mathrm{out}}n_{\mathrm{in}}$ and $R_{\mathrm{full}} = n_{\mathrm{out}}(n_{\mathrm{in}} - k_{\mathrm{in}}) + (n_{\mathrm{out}} - k_{\mathrm{out}})k_{\mathrm{in}}$. This expression isolates the graph-dependent neural computation; the FLOPs reported in the main efficiency comparison additionally include local inner-code processing.

B Evaluation Protocol Details

BER is measured over the complete reconstructed binary concatenated word, while BLER is the fraction of transmitted concatenated words for which $\widehat{\mathbf{x}} \neq \mathbf{x}$. With unit-energy BPSK and overall rate $R = k_{\mathrm{out}}k_{\mathrm{in}}/(n_{\mathrm{out}}n_{\mathrm{in}})$, the AWGN variance is $\sigma^2 = 1/(2R\,10^{(E_b/N_0)_{\mathrm{dB}}/10})$. For each configuration, all methods use the same constituent codes, extension-field basis, and binary coordinate order. Neural models are optimized with Adam from an initial learning rate of 10^{-4} using cosine decay to 5×10^{-7}, with random seed 42; the checkpoint after 1000 training epochs is used for evaluation. Testing uses batches of 2048 codewords. At each SNR, evaluation covers at least 100,000 codewords and continues until more than 100 frame errors have been observed, subject to a maximum of 10^9 codewords. The remaining training SNR, batch-size, model-width, and depth settings are specified in Section 4.1.

Aggregate comparisons with EfficientMPT count each code once, using the default interface where available and the higher-dimensional interfaces otherwise. BER ratios are averaged geometrically; FLOPs and memory reductions are averaged arithmetically.

C Replaceable Local Interfaces and Higher-Dimensional Inner Codes

We evaluate Chase-style soft estimation for BCH and linear-trellis soft-output estimation for RM as alternatives to the direct max-log implementation. Each supplies tentative outer bits and reliability under the coordinate convention in Appendix A.8. For a fixed code, this replacement preserves the outer graph and residual task. We also apply these interfaces to inner-code dimensions 12 and 16, using the same neural architecture on the corresponding outer graphs.

Table 4: Local-interface replacement and higher-dimensional inner-code results. For each model, the first three columns report $-\ln(\mathrm{BER})$ at SNRs of 4, 5, and 6 dB, followed by FLOPs in G/codeword and peak training memory in MiB. The CR-CCM interface is Chase for BCH and a linear trellis for RM. Boldface marks the best rounded decoding results and the CR-CCM efficiency values.

Concatenated code	Alternative interface	EfficientMPT					CR-CCM-EfficientMPT				
		4	5	6	FLOPs (G)	Memory (MiB)	4	5	6	FLOPs (G)	Memory (MiB)
NB-LDPC$(63, 55; 6)$ + BCH$(14, 6)$	Chase	6.09	8.07	10.41	4.72	93	**6.40**	**8.53**	**10.91**	**0.98**	**51.03**
RS$(32, 26; 6)$ + RM$(32, 6)$	Linear trellis	5.33	6.87	**8.90**	7.05	113	**5.35**	**6.89**	8.86	**0.51**	**43.09**
RS$(63, 55; 6)$ + RM$(32, 6)$	Linear trellis	5.52	7.09	9.12	19.32	195	**5.79**	**7.44**	**9.58**	**0.98**	**51.07**
RS$(80, 68; 12)$ + BCH$(22, 12)$	Chase	4.13	5.21	6.58	11.38	148	**5.20**	**6.79**	**8.85**	**2.85**	**79**
RS$(31, 25; 16)$ + RM$(32, 16)$	Linear trellis	5.52	7.44	10.05	5.38	99	**5.79**	**7.80**	**10.45**	**1.44**	**59**

Table 4 shows that both interfaces support neural residual correction on the reduced outer graph. The Chase-based instances improve upon full-binary EfficientMPT at all three SNRs, while the trellis-based instances remain close to or improve upon it. All use substantially fewer FLOPs and less memory. The examples with inner-code dimensions 12 and 16 also improve BER and efficiency. These results support replacing the local estimator and extending the framework to higher-dimensional inner codes without changing the neural architecture.

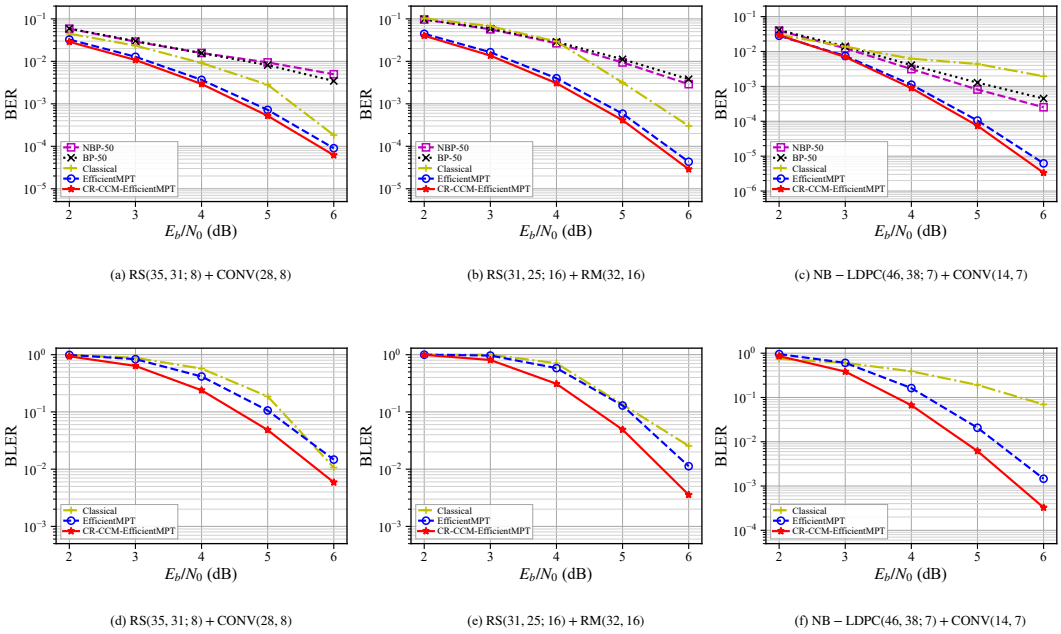
D Additional Neural Decoding Results

Table 5 provides additional neural decoding results.

Table 5: Additional neural decoding results. Entries are $-\ln(\mathrm{BER})$ at the indicated E_b/N_0 in dB, and higher is better. "–" denotes an unavailable result, and boldface marks the best rounded result.

Concatenated code	NBP-50			EfficientMPT			CR-CCM-EfficientMPT		
	4	5	6	4	5	6	4	5	6
$\mathrm{RS}(31,23;5)+\mathrm{BCH}(15,5)$	3.54	4.02	4.53	**4.47**	**5.60**	7.04	**4.47**	**5.60**	**7.05**
$\mathrm{RS}(32,26;6)+\mathrm{CONV}(12,6)$	4.26	5.37	6.70	**4.61**	**5.73**	**7.13**	**4.61**	**5.73**	**7.13**
$\mathrm{RS}(32,26;6)+\mathrm{RM}(32,6)$	3.22	3.36	3.55	5.33	6.87	8.90	**5.36**	**6.90**	**8.97**
$\mathrm{RS}(63,47;6)+\mathrm{BCH}(12,6)$	3.71	4.36	5.03	4.25	5.27	6.58	**4.26**	**5.30**	**6.62**
$\mathrm{RS}(63,47;6)+\mathrm{CONV}(12,6)$	3.76	4.47	5.21	4.24	5.25	6.53	**4.26**	**5.28**	**6.58**
$\mathrm{RS}(63,55;6)+\mathrm{BCH}(12,6)$	4.28	5.14	6.25	**4.94**	6.14	7.68	**4.94**	**6.16**	**7.72**
$\mathrm{RS}(63,55;6)+\mathrm{CONV}(12,6)$	4.36	5.30	6.50	4.92	6.11	7.63	**4.93**	**6.12**	**7.64**
$\mathrm{RS}(63,55;6)+\mathrm{GOLAY}(12,6)$	4.49	5.60	7.01	4.93	6.15	7.70	**4.94**	**6.17**	**7.73**
$\mathrm{RS}(63,55;6)+\mathrm{POLAR}(16,6)$	–	–	–	**3.28**	**3.57**	**3.91**	**3.28**	**3.57**	**3.91**
$\mathrm{RS}(46,34;7)+\mathrm{EXT\text{-}HAM}(12,7)$	3.90	4.96	6.19	**4.25**	**5.33**	**6.73**	**4.25**	**5.33**	**6.73**
$\mathrm{RS}(46,34;7)+\mathrm{GOLAY}(14,7)$	3.85	4.71	5.63	4.34	5.43	6.84	**4.37**	**5.48**	**6.91**
$\mathrm{RS}(46,38;7)+\mathrm{BCH}(15,7)$	4.71	6.18	**8.11**	4.91	6.23	7.91	**4.95**	**6.29**	7.96
$\mathrm{RS}(34,28;8)+\mathrm{EXT\text{-}HAM}(13,8)$	4.38	5.73	7.35	**4.77**	**6.05**	7.71	**4.77**	**6.05**	**7.72**
$\mathrm{RS}(34,28;8)+\mathrm{PRODUCT}(15,8)$	**4.65**	**6.20**	**8.11**	4.63	5.79	7.26	4.63	5.79	7.26
$\mathrm{RS}(107,99;8)+\mathrm{POLAR}(16,8)$	**4.18**	**4.60**	**4.98**	3.45	3.83	4.29	3.45	3.83	4.29
$\mathrm{NB\text{-}LDPC}(63,55;6)+\mathrm{BCH}(12,6)$	5.42	6.84	8.17	6.62	8.90	11.76	**6.67**	**9.09**	**12.12**
$\mathrm{NB\text{-}LDPC}(63,55;6)+\mathrm{BCH}(31,6)$	3.98	4.69	5.37	5.75	7.38	9.54	**8.02**	**10.94**	**14.20**
$\mathrm{NB\text{-}LDPC}(63,55;6)+\mathrm{CONV}(12,6)$	5.57	7.07	8.45	6.75	9.15	12.16	**6.85**	**9.42**	**12.65**
$\mathrm{NB\text{-}LDPC}(46,38;7)+\mathrm{BCH}(14,7)$	5.53	6.76	7.82	6.88	9.24	12.11	**7.04**	**9.54**	**12.54**

For $\mathrm{RS}(34,28;8)+\mathrm{EXT\text{-}HAM}(13,8)$, the classical reference gives $-\ln(\mathrm{BER})$ values of 3.86, 6.39, and 11.04 at 4, 5, and 6 dB, respectively.



(a) RS(35, 31; 8) + CONV(28, 8) (b) RS(31, 25; 16) + RM(32, 16) (c) NB – LDPC(46, 38; 7) + CONV(14, 7)

(d) RS(35, 31; 8) + CONV(28, 8) (e) RS(31, 25; 16) + RM(32, 16) (f) NB – LDPC(46, 38; 7) + CONV(14, 7)

Figure 6: Additional BER and BLER curves. The top row compares all five decoders, while the bottom row compares the classical baseline, EfficientMPT, and CR-CCM-EfficientMPT. The concatenated-code configuration is given below each panel, and all vertical axes are logarithmic.

The broader comparison follows the trend observed in the main experiments: CR-CCM-EfficientMPT generally matches or improves the full-binary EfficientMPT reference. Together with Tables 1 and 4, these results give an average gain of 0.603 in $-\ln(\mathrm{BER})$ at $\mathrm{SNR}=6$ dB, corresponding to a $1.83\times$ geometric-mean BER reduction.

Figure 6 shows that the separation is sustained from 2 to 6 dB. The largest tabulated separation occurs for NB-LDPC$(63, 55; 6) + \mathrm{BCH}(31, 6)$ at $\mathrm{SNR} = 6$ dB. Here, $-\ln(\mathrm{BER})$ increases from 9.54 to 14.20, corresponding to an approximately $106\times$ BER reduction.

E Additional Efficiency Results

Table 6 provides additional FLOPs and memory results. Each cell reports the measured value and, in parentheses, its percentage relative to ECCT; lower is better. The reported CR-CCM-EfficientMPT FLOPs include local inner-code processing. Together with Tables 2 and 4, these measurements give average FLOPs and memory reductions of 75.8% and 42.2%, respectively, relative to EfficientMPT.

Table 6: Additional efficiency results. Memory usage is the peak allocated memory during training.

Concatenated code	FLOPs (G/codeword)				Peak training memory (MiB)			
	ECCT	CrossMPT	EfficientMPT	CR-CCM-EfficientMPT	ECCT	CrossMPT	EfficientMPT	CR-CCM-EfficientMPT
RS$(31, 25; 5) + \mathrm{BCH}(15, 5)$	3.89 (100%)	2.87 (73.8%)	2.27 (58.3%)	**0.42** (10.7%)	395 (100%)	219 (55.6%)	67 (16.9%)	**41** (10.5%)
RS$(34, 28; 8) + \text{EXT-HAM}(13, 8)$	2.90 (100%)	2.15 (74.2%)	1.75 (60.4%)	**0.74** (25.4%)	281 (100%)	155 (55.2%)	61 (21.5%)	**47** (16.6%)
RS$(31, 23; 5) + \mathrm{BCH}(15, 5)$	3.97 (100%)	2.93 (73.8%)	2.31 (58.1%)	**0.44** (11.2%)	403 (100%)	224 (55.5%)	67 (16.6%)	**42** (10.3%)
RS$(32, 26; 6) + \mathrm{CONV}(12, 6)$	2.60 (100%)	1.98 (76.4%)	1.62 (62.2%)	**0.52** (19.9%)	248 (100%)	144 (58.0%)	59 (23.7%)	**43** (17.3%)
RS$(32, 26; 6) + \mathrm{RM}(32, 6)$	15.47 (100%)	9.93 (64.2%)	7.05 (45.6%)	**0.52** (3.3%)	1923 (100%)	959 (49.9%)	113 (5.9%)	**43** (2.2%)
RS$(63, 47; 6) + \mathrm{BCH}(12, 6)$	7.55 (100%)	5.11 (67.6%)	3.85 (51.0%)	**1.15** (15.2%)	852 (100%)	430 (50.5%)	84 (9.9%)	**53** (6.2%)
RS$(63, 47; 6) + \mathrm{CONV}(12, 6)$	7.55 (100%)	5.11 (67.6%)	3.85 (51.0%)	**1.15** (15.2%)	852 (100%)	430 (50.5%)	84 (9.9%)	**53** (6.2%)
RS$(63, 55; 6) + \mathrm{BCH}(12, 6)$	7.08 (100%)	4.77 (67.3%)	3.63 (51.2%)	**0.98** (13.9%)	791 (100%)	395 (49.9%)	82 (10.4%)	**51** (6.4%)
RS$(63, 55; 6) + \mathrm{CONV}(12, 6)$	7.08 (100%)	4.77 (67.3%)	3.63 (51.2%)	**0.98** (13.9%)	791 (100%)	395 (49.9%)	82 (10.4%)	**51** (6.4%)
RS$(63, 55; 6) + \mathrm{GOLAY}(12, 6)$	7.08 (100%)	4.77 (67.3%)	3.63 (51.2%)	**0.98** (13.9%)	791 (100%)	395 (49.9%)	82 (10.4%)	**51** (6.4%)
RS$(63, 55; 6) + \mathrm{POLAR}(16, 6)$	12.72 (100%)	8.18 (64.4%)	5.92 (46.6%)	**0.98** (7.7%)	1544 (100%)	759 (49.2%)	104 (6.7%)	**51** (3.3%)
RS$(46, 34; 7) + \text{EXT-HAM}(12, 7)$	4.35 (100%)	3.11 (71.5%)	2.45 (56.4%)	**0.97** (22.3%)	450 (100%)	238 (52.9%)	69 (15.4%)	**50** (11.2%)
RS$(46, 34; 7) + \mathrm{GOLAY}(14, 7)$	5.87 (100%)	4.09 (69.7%)	3.14 (53.6%)	**0.97** (16.5%)	636 (100%)	330 (51.9%)	77 (12.1%)	**50** (7.9%)
RS$(46, 38; 7) + \mathrm{BCH}(15, 7)$	6.44 (100%)	4.43 (68.7%)	3.38 (52.5%)	**0.88** (13.6%)	709 (100%)	362 (51.1%)	80 (11.2%)	**49** (6.9%)
RS$(34, 28; 8) + \mathrm{PRODUCT}(15, 8)$	3.83 (100%)	2.78 (72.6%)	2.21 (57.7%)	**0.74** (19.2%)	388 (100%)	208 (53.8%)	66 (17.1%)	**47** (12.1%)
RS$(107, 99; 8) + \mathrm{POLAR}(16, 8)$	27.50 (100%)	15.90 (57.8%)	10.94 (39.8%)	**2.17** (7.9%)	3658 (100%)	1659 (45.4%)	145 (4.0%)	**71** (1.9%)
NB-LDPC$(63, 55; 6) + \mathrm{GOLAY}(12, 6)$	7.08 (100%)	4.77 (67.3%)	3.63 (51.2%)	**0.98** (13.9%)	791 (100%)	395 (49.9%)	82 (10.4%)	**51** (6.4%)
NB-LDPC$(46, 38; 7) + \mathrm{CONV}(14, 7)$	5.62 (100%)	3.91 (69.5%)	3.02 (53.7%)	**0.88** (15.6%)	605 (100%)	312 (51.5%)	76 (12.5%)	**49** (8.1%)
NB-LDPC$(63, 55; 6) + \mathrm{BCH}(12, 6)$	7.08 (100%)	4.77 (67.3%)	3.63 (51.2%)	**0.98** (13.9%)	791 (100%)	395 (49.9%)	82 (10.4%)	**51** (6.4%)
NB-LDPC$(63, 55; 6) + \mathrm{BCH}(31, 6)$	47.74 (100%)	27.93 (58.5%)	18.29 (38.3%)	**0.98** (2.1%)	6649 (100%)	3191 (48.0%)	189 (2.8%)	**51** (0.8%)
NB-LDPC$(63, 55; 6) + \mathrm{CONV}(12, 6)$	7.08 (100%)	4.77 (67.3%)	3.63 (51.2%)	**0.98** (13.9%)	791 (100%)	395 (49.9%)	82 (10.4%)	**51** (6.4%)
NB-LDPC$(46, 38; 7) + \mathrm{BCH}(14, 7)$	5.62 (100%)	3.91 (69.5%)	3.02 (53.7%)	**0.88** (15.6%)	605 (100%)	312 (51.5%)	76 (12.5%)	**49** (8.1%)

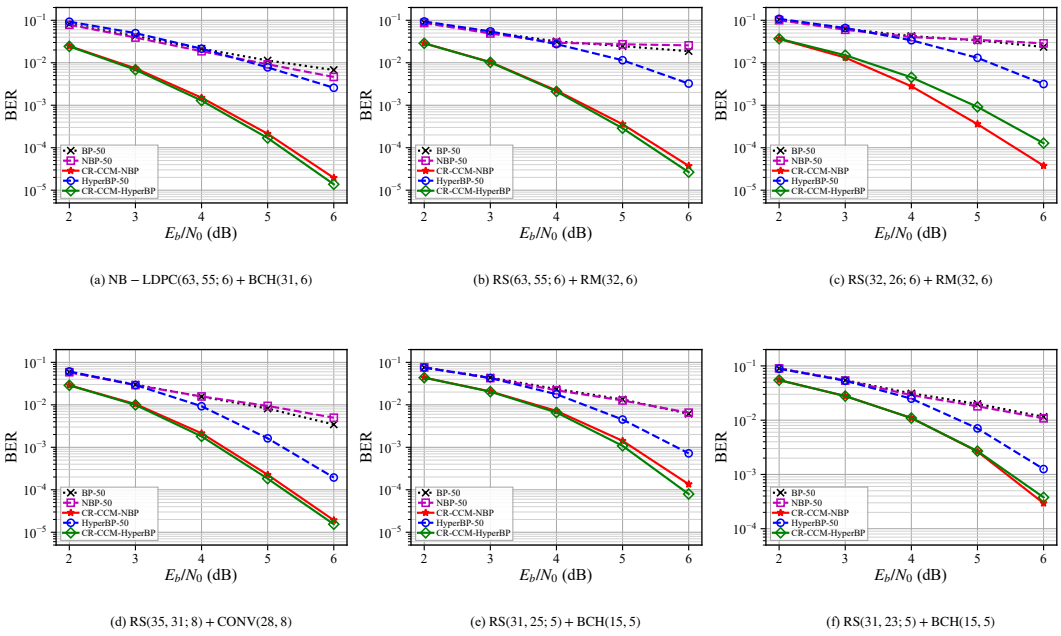
The dense EfficientMPT implementation shows large savings when many repeated inner-code variables and checks are removed. For example, on NB-LDPC$(63, 55; 6) + \mathrm{BCH}(31, 6)$, the FLOPs decrease from 18.29 to 0.98 G/codeword, and memory usage decreases from 189 to 51 MiB. Savings also remain visible for shorter inner codes. This trend is consistent with the smaller graph in Equation (6) and the dense-operation analysis in Appendix A.9. Together with the BER results in Appendix D, it shows that the reduced task can lower the computational cost while maintaining or improving decoding performance.

F BP-BASED INSTANTIATIONS: DECODING PERFORMANCE

We further instantiate CR-CCM with the NBP and HyperBP backbones. Detailed decoding results are provided in Table 7.

Table 7: Decoding performance of NBP, HyperBP, and their CR-CCM counterparts. Entries are $-\ln(\mathrm{BER})$ at the indicated E_b/N_0 in dB, and higher is better. Boldface marks the better rounded result within each backbone pair.

Concatenated code	NBP			CR-CCM-NBP			HyperBP			CR-CCM-HyperBP		
	4	5	6	4	5	6	4	5	6	4	5	6
$\mathrm{RS}(31,25;5)+\mathrm{BCH}(15,5)$	3.82	4.37	5.03	**4.93**	**6.56**	**8.91**	4.03	5.40	7.24	**5.02**	**6.83**	**9.44**
$\mathrm{RS}(63,55;6)+\mathrm{BCH}(14,6)$	4.47	5.81	7.68	**5.09**	**6.41**	**8.07**	4.61	6.12	7.90	**5.09**	**6.41**	**8.07**
$\mathrm{RS}(35,31;8)+\mathrm{CONV}(28,8)$	4.15	4.66	5.30	**6.15**	**8.40**	**10.86**	4.68	6.42	8.55	**6.31**	**8.61**	**11.08**
$\mathrm{RS}(107,99;8)+\mathrm{LDPC}(16,8)$	4.96	6.10	7.64	**5.25**	**6.55**	**8.18**	3.66	4.23	4.93	**5.25**	**6.55**	**8.18**
$\mathrm{RS}(31,23;5)+\mathrm{BCH}(15,5)$	3.54	4.02	4.53	**4.51**	**5.94**	**8.13**	3.69	4.95	6.68	**4.51**	**5.92**	**7.88**
$\mathrm{RS}(32,26;6)+\mathrm{CONV}(12,6)$	4.26	5.37	6.70	**4.62**	**5.72**	**7.15**	4.40	**5.92**	**7.97**	**4.62**	5.72	7.15
$\mathrm{RS}(32,26;6)+\mathrm{RM}(32,6)$	3.22	3.36	3.55	**5.88**	**7.93**	**10.18**	3.39	4.34	5.76	**5.40**	**7.00**	**8.97**
$\mathrm{RS}(63,47;6)+\mathrm{BCH}(12,6)$	3.71	4.36	5.03	**4.27**	**5.30**	**6.61**	3.80	4.82	6.43	**4.27**	**5.30**	**6.61**
$\mathrm{RS}(63,47;6)+\mathrm{CONV}(12,6)$	3.76	4.47	5.21	**4.26**	**5.28**	**6.57**	3.87	4.96	**6.68**	**4.26**	**5.28**	6.57
$\mathrm{RS}(63,55;6)+\mathrm{BCH}(12,6)$	4.28	5.14	6.25	**4.94**	**6.17**	**7.71**	4.45	5.92	**7.93**	**4.94**	**6.17**	7.71
$\mathrm{RS}(63,55;6)+\mathrm{CONV}(12,6)$	4.36	5.30	6.50	**4.93**	**6.13**	**7.67**	4.52	6.04	**8.12**	**4.93**	**6.13**	7.67
$\mathrm{RS}(63,55;6)+\mathrm{GOLAY}(12,6)$	4.49	5.60	7.01	**4.95**	**6.17**	**7.73**	4.63	**6.22**	**8.36**	**4.95**	6.17	7.73
$\mathrm{RS}(63,55;6)+\mathrm{RM}(32,6)$	3.52	3.60	3.66	**6.11**	**7.94**	**10.20**	3.59	4.47	5.73	**6.16**	**8.16**	**10.54**
$\mathrm{RS}(46,34;7)+\mathrm{GOLAY}(14,7)$	3.85	4.71	5.63	**4.36**	**5.49**	**6.91**	3.94	5.28	**7.22**	**4.36**	**5.49**	6.91
$\mathrm{RS}(107,99;8)+\mathrm{POLAR}(16,8)$	4.18	4.60	4.98	**5.08**	**6.30**	**7.85**	4.66	5.95	**7.89**	**5.08**	**6.30**	7.85
$\text{NB-LDPC}(63,55;6)+\mathrm{BCH}(14,6)$	5.36	6.94	8.64	**5.54**	**7.22**	**9.28**	5.18	6.71	8.35	**5.88**	**7.72**	**9.92**
$\text{NB-LDPC}(63,55;6)+\mathrm{GOLAY}(12,6)$	**5.75**	**7.34**	8.81	5.28	6.84	**8.86**	5.53	7.20	9.02	**5.67**	**7.43**	**9.59**
$\text{NB-LDPC}(46,38;7)+\mathrm{CONV}(14,7)$	**5.77**	**7.12**	8.28	5.34	6.99	**8.93**	5.66	7.30	9.05	**5.80**	**7.60**	**9.78**
$\text{NB-LDPC}(107,99;8)+\mathrm{CONV}(16,8)$	5.10	6.51	7.84	**5.77**	**7.50**	**9.77**	5.05	6.70	8.57	**5.71**	**7.27**	**9.21**
$\text{NB-LDPC}(63,55;6)+\mathrm{BCH}(12,6)$	**5.42**	**6.84**	8.17	5.15	6.62	**8.45**	5.15	6.72	8.55	**5.56**	**7.27**	**9.40**
$\text{NB-LDPC}(63,55;6)+\mathrm{BCH}(31,6)$	3.98	4.69	5.37	**6.50**	**8.45**	**10.84**	3.84	4.86	5.96	**6.65**	**8.69**	**11.19**
$\text{NB-LDPC}(63,55;6)+\mathrm{CONV}(12,6)$	**5.57**	**7.07**	8.45	5.24	6.78	**8.76**	5.35	6.92	8.67	**5.64**	**7.39**	**9.56**
$\text{NB-LDPC}(46,38;7)+\mathrm{BCH}(14,7)$	**5.53**	6.76	7.82	5.36	**7.06**	**9.08**	5.41	7.03	8.77	**5.77**	**7.60**	**9.85**



(a) NB − LDPC(63, 55; 6) + BCH(31, 6) (b) RS(63, 55; 6) + RM(32, 6) (c) RS(32, 26; 6) + RM(32, 6)

(d) RS(35, 31; 8) + CONV(28, 8) (e) RS(31, 25; 5) + BCH(15, 5) (f) RS(31, 23; 5) + BCH(15, 5)

Figure 7: Representative BER curves for BP-50, NBP-50, CR-CCM-NBP, HyperBP-50, and CR-CCM-HyperBP. The concatenated-code configuration is given below each panel, and all vertical axes are logarithmic.

Figure 7 illustrates the BER gains for both BP-based instances over 2–6 dB. At $\mathrm{SNR} = 6\ \mathrm{dB}$, the mean gains in $-\ln(\mathrm{BER})$ relative to the corresponding full-binary backbones are 2.11 for NBP and 1.15 for HyperBP. The median BER reductions are $3.60\times$ and $2.09\times$, respectively. For example, on $\mathrm{RS}(32, 26; 6) + \mathrm{RM}(32, 6)$, NBP improves from 3.55 to 10.18 in $-\ln(\mathrm{BER})$. HyperBP improves from 5.96 to 11.19 on $\mathrm{NB\text{-}LDPC}(63, 55; 6) + \mathrm{BCH}(31, 6)$.

The gain varies with the code and backbone. Overall, the paired comparisons support using the same local interface and residual construction with different neural architectures. Each backbone is adapted to the reduced graph and prediction target, and trained for that task. The efficiency results in Appendix G further show the computational benefit of these BP-based instances.

G BP-Based Instantiations: Efficiency

Table 8 reports FLOPs and peak training memory usage for each backbone and its CR-CCM counterpart, measured after fresh initialization with batch size one. Each cell gives the measured value and its percentage relative to the corresponding backbone; lower is better. The CR-CCM measurements include both local inner-code processing and the reduced neural decoder.

Table 8: Efficiency of NBP, HyperBP, and their CR-CCM counterparts.

Concatenated code	FLOPs (G/codeword)				Memory usage (MiB)			
	NBP	CR-CCM-NBP	HyperBP	CR-CCM-HyperBP	NBP	CR-CCM-NBP	HyperBP	CR-CCM-HyperBP
$\mathrm{RS}(31,25;5)+\mathrm{BCH}(15,5)$	1.37 (100%)	**0.58** (42.1%)	0.18 (100%)	**0.10** (52.3%)	68 (100%)	**42** (61.9%)	23 (100%)	**20** (85.9%)
$\mathrm{RS}(63,55;6)+\mathrm{BCH}(14,6)$	12.93 (100%)	**8.51** (65.9%)	0.70 (100%)	**0.56** (79.1%)	524 (100%)	**361** (69.0%)	38 (100%)	**32** (82.8%)
$\mathrm{RS}(35,31;8)+\mathrm{CONV}(28,8)$	5.29 (100%)	**1.64** (31.1%)	0.42 (100%)	**0.17** (41.7%)	224 (100%)	**85** (37.9%)	30 (100%)	**22** (74.1%)
$\mathrm{RS}(107,99;8)+\mathrm{LDPC}(16,8)$	93.74 (100%)	**73.94** (78.9%)	2.63 (100%)	**2.21** (83.9%)	3672 (100%)	**2913** (79.3%)	87 (100%)	**71** (82.5%)
$\mathrm{RS}(31,23;5)+\mathrm{BCH}(15,5)$	2.03 (100%)	**1.03** (50.5%)	0.27 (100%)	**0.16** (57.9%)	96 (100%)	**62** (64.9%)	25 (100%)	**21** (84.3%)
$\mathrm{RS}(32,26;6)+\mathrm{CONV}(12,6)$	1.80 (100%)	**1.12** (62.2%)	0.22 (100%)	**0.16** (71.7%)	85 (100%)	**65** (76.1%)	24 (100%)	**21** (88.8%)
$\mathrm{RS}(32,26;6)+\mathrm{RM}(32,6)$	4.97 (100%)	**1.12** (22.4%)	0.51 (100%)	**0.16** (31.5%)	215 (100%)	**65** (30.0%)	31 (100%)	**21** (68.4%)
$\mathrm{RS}(63,47;6)+\mathrm{BCH}(12,6)$	41.28 (100%)	**34.19** (82.8%)	2.27 (100%)	**1.92** (84.5%)	1667 (100%)	**1394** (83.7%)	72 (100%)	**60** (83.9%)
$\mathrm{RS}(63,47;6)+\mathrm{CONV}(12,6)$	41.02 (100%)	**34.19** (83.3%)	2.20 (100%)	**1.92** (87.3%)	1657 (100%)	**1394** (84.2%)	73 (100%)	**60** (83.2%)
$\mathrm{RS}(63,55;6)+\mathrm{BCH}(12,6)$	12.22 (100%)	**8.51** (69.7%)	0.76 (100%)	**0.56** (73.7%)	501 (100%)	**361** (72.1%)	39 (100%)	**32** (80.9%)
$\mathrm{RS}(63,55;6)+\mathrm{CONV}(12,6)$	12.08 (100%)	**8.51** (70.5%)	0.72 (100%)	**0.56** (77.8%)	495 (100%)	**361** (73.0%)	38 (100%)	**32** (82.7%)
$\mathrm{RS}(63,55;6)+\mathrm{GOLAY}(12,6)$	11.94 (100%)	**8.51** (71.3%)	0.71 (100%)	**0.56** (78.2%)	490 (100%)	**361** (73.7%)	38 (100%)	**32** (82.8%)
$\mathrm{RS}(63,55;6)+\mathrm{RM}(32,6)$	27.34 (100%)	**8.52** (31.1%)	1.40 (100%)	**0.56** (39.8%)	1101 (100%)	**361** (32.8%)	55 (100%)	**32** (57.2%)
$\mathrm{RS}(46,34;7)+\mathrm{GOLAY}(14,7)$	21.52 (100%)	**17.17** (79.8%)	1.48 (100%)	**1.21** (82.2%)	882 (100%)	**716** (81.1%)	54 (100%)	**46** (84.4%)
$\mathrm{RS}(107,99;8)+\mathrm{POLAR}(16,8)$	99.05 (100%)	**73.94** (74.7%)	2.71 (100%)	**2.21** (81.6%)	3878 (100%)	**2913** (75.1%)	88 (100%)	**71** (80.7%)
$\text{NB-LDPC}(63,55;6)+\mathrm{BCH}(14,6)$	1.14 (100%)	**0.15** (13.4%)	0.07 (100%)	**0.03** (34.6%)	55 (100%)	**23** (42.5%)	21 (100%)	**18** (86.6%)
$\text{NB-LDPC}(63,55;6)+\mathrm{GOLAY}(12,6)$	0.86 (100%)	**0.15** (17.7%)	0.08 (100%)	**0.03** (30.4%)	43 (100%)	**23** (53.3%)	20 (100%)	**18** (87.8%)
$\text{NB-LDPC}(46,38;7)+\mathrm{CONV}(14,7)$	0.75 (100%)	**0.16** (21.3%)	0.08 (100%)	**0.03** (36.6%)	39 (100%)	**24** (60.2%)	20 (100%)	**18** (89.1%)
$\text{NB-LDPC}(107,99;8)+\mathrm{CONV}(16,8)$	7.17 (100%)	**1.11** (15.5%)	0.26 (100%)	**0.08** (29.3%)	290 (100%)	**62** (21.2%)	29 (100%)	**21** (71.8%)
$\text{NB-LDPC}(63,55;6)+\mathrm{BCH}(12,6)$	0.94 (100%)	**0.15** (16.3%)	0.08 (100%)	**0.03** (31.0%)	47 (100%)	**23** (49.5%)	21 (100%)	**18** (87.4%)
$\text{NB-LDPC}(63,55;6)+\mathrm{BCH}(31,6)$	6.87 (100%)	**0.15** (2.2%)	0.37 (100%)	**0.03** (6.9%)	282 (100%)	**23** (8.2%)	30 (100%)	**18** (60.3%)
$\text{NB-LDPC}(63,55;6)+\mathrm{CONV}(12,6)$	0.90 (100%)	**0.15** (17.0%)	0.08 (100%)	**0.03** (29.8%)	45 (100%)	**23** (51.5%)	21 (100%)	**18** (87.4%)
$\text{NB-LDPC}(46,38;7)+\mathrm{BCH}(14,7)$	0.85 (100%)	**0.16** (18.7%)	0.09 (100%)	**0.03** (34.3%)	43 (100%)	**24** (54.3%)	21 (100%)	**18** (88.1%)

Relative to NBP, FLOPs and memory usage decrease by 54.9% and 41.9% on average and by up to 97.8% and 91.8%. Relative to HyperBP, the average reductions are 45.4% and 19.1%, with maxima of 93.1% and 42.8%. Thus, the efficiency benefit of the reduced task is also observed in these BP-based instances.