# RETHINKING DIFFUSION FOR NEURAL CHANNEL DECODING: ALIGNED TRAINING AND HETEROGENEOUS INFERENCE

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Neural channel decoders offer a common approach to decoding different families of linear block codes, but improving error correction often comes at the cost of slower inference. Diffusion decoders make this trade-off particularly apparent: they rely on repeated model evaluations, while existing training objectives do not always match the dynamics used at inference. We revisit diffusion decoding by aligning the training target with probability-flow dynamics and mapping the channel noise level directly to diffusion time. This yields DEER, a decoder that supports a fixed inference schedule and improves decoding performance in both one-step and multi-step settings. To reduce the cost of each step, we develop FAWN, a lightweight decoder with computation that scales linearly with the input sequence length and the number of nonzero entries in the parity-check matrix. We further observe that the high-capacity decoder is most useful in the noisier stages of a decoding trajectory, whereas lightweight decoders become competitive as the noise decreases. This motivates heterogeneous inference, which assigns DEER and FAWN to different stages of the same trajectory. Across several code families, the resulting decoders improve the measured bit-error-rate–latency trade-off, with heterogeneous inference matching or exceeding the performance of the high-capacity decoder at lower latency.

## 1 INTRODUCTION

Beyond earlier neural decoders(Nachmani et al., 2016; 2018; Gruber et al., 2017; Lugosch & Gross, 2017; Bennatan et al., 2018) that augment traditional decoding algorithms, more general neural decoders(Choukroun & Wolf, 2023; 2022; Zhang et al., 2024; Park et al., 2026; 2025; Lei et al., 2025; Zheng et al., 2026) based on Transformer architectures have recently emerged, featuring decoding through the attention mechanism for handling channel observations. Diffusion-based decoders(Choukroun & Wolf, 2023; Zheng et al., 2026; Lei et al., 2025) were also proposed to iteratively recover the transmitted codeword.

The design of diffusion-based channel decoders has unresolved gaps in their formulation, weakening their theoretical justification. Lei et al. (2025) addressed this loss function gap by bounding the actual loss with a surrogate loss, which is still not exact. In practice, the neural channel decoders require high computation/memory resources, with the complexity growing nonlinearly as code length extends, hindering their application to longer codes. In short codes, they were outpaced by classical decoders. Some diffusion decoders are even heavier, as pointed out by Lei et al. (2025), with a growing average number of steps when code length increases.

In this work, we revisit the formulation for diffusion neural decodings and redesign both its training and inference schedules. This new design on the CrossMPT(Park et al., 2025)-based model DEER has demonstrated improved performance, especially for the low SNR range where the decoding is harder, in both multiple- and single-step diffusion setups. Our formulation also makes multi-step decoding faster by fixing inference steps and therefore using a fixed number of diffusion steps. The actual runtime result is shown in Appendix E.

Moreover, this pipeline was incorporated with the EfficientMPT (Park et al., 2026) backbone and underwent architectural pruning and modification, resulting in a series of models named FAWN



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.

with very light computation cost and latency, at a cost of slightly degraded decoding performance. We then incorporate FAWN with DEER into a hybrid diffusion framework that utilizes the models' advantages during different decoding stages, as observed from our experiments. The hybrid method shows promising results by exhibiting a FAWN-equivalent computation cost and higher overall error recovery comparable to or even exceeding other methods.

In short, we i) propose a cleaner and simpler diffusion pipeline for neural decoders, contributing to a new model DEER obtaining competitive performance for low SNR regions across code families; ii) develop an architecture-optimized lightweight model FAWN that achieves comparable performance; iii) combine DEER and FAWN around the diffusion steps for a heterogeneous inference architecture that further improves the performance-latency trade-off.

## 2 Background and Related Work

**Neural Channel Decoding** Channel coding has been extensively studied in the field of communication, such as OSD (Fossorier & Lin, 1995), SCL (Tal & Vardy, 2015) for polar codes, and BP algorithms for LDPC codes(Kschischang et al., 2001).

Following the rise of research in deep learning, Nachmani et al. (2016); Gruber et al. (2017); Lugosch & Gross (2017); Nachmani et al. (2018) proposed to apply deep learning to traditional BP algorithms. The emergence of attention (Vaswani et al., 2017) has given rise to new neural decoding approaches (Bennatan et al., 2018; Choukroun & Wolf, 2023), which was further developed by Choukroun & Wolf (2022); Zhang et al. (2024); Park et al. (2026; 2025); Lei et al. (2025).

Computation cost increases for neural decoders, with attempts to lighten the cost, of which Levy et al. (2024) use ternary weight quantization, Zhang et al. (2024) utilize the quasi-cyclic structures for the parity-check matrix, Park et al. (2026) use near-linear complexity attention design, Li et al. (2026) use the product structure of product code, and Lei et al. (2025) with a 1 step diffusion framework. Apart from those, Cho et al. (2026) prune the attention heads using spectral information.

**Diffusion Models** The discrete (Song et al., 2021a; Ho et al., 2020) and score-based (Song et al., 2021b) diffusion models have demonstrated remarkable capabilities in generation and denoising, and are unified by the Stochastic Differential Equations(SDE) framework. The probability flow in SDE is also closely related to the topics of optimal transport, where the ML decoding can be viewed as one special case in optimal transport. Inspired by connecting the diffusion SDE to the Schrödinger Bridge by Liu et al. (2023), and by Jiang et al. (2025); Yue et al. (2023), a bridge for channel noise is incorporated in our formulation.

## 3 Preliminaries

### 3.1 Channel coding and Linear Codes

Channel coding refers to transmission of information reliably through noisy channels. With a message $m \in \{0,1\}^K$, it is first encoded into a length-$N$ codeword $C = Gm \in \{0,1\}^N$ by the generator matrix $G$. Then, for Additive White Gaussian Noise channels, codeword $C$ is modulated into a signal $X \in \{\pm 1\}^N$, sent through the channel, from which the receiver obtains observations $Y = X + n$, $n \sim \mathcal{N}(0, \sigma^2 \mathrm{I}_N)$. The noise level $\sigma$ is determined by the channel's signal-to-noise ratio (SNR), often measured by $-10\log_{10}\left(2 \cdot \frac{K}{N}\sigma^2\right)$ in dB.

Upon receiving $Y$, log-likelihood ratios (LLRs) for each symbol bit are estimated, then fed into a decoder utilizing the code redundancy characterized by check matrix $H$. For a corrupted codeword $m'$, the syndrome $s = m'H^\mathsf{T}$ holds sufficient information about its error patterns. The syndrome is calculated by $s(Y) = H\frac{1-\mathrm{sign}(Y)}{2}$ for an observation $Y$.

Choukroun & Wolf (2022) have proved that the magnitude of LLR along with the syndromes are sufficient for decoding. Thus, our models are essentially

$$\hat{X} = \mathrm{sign}(Y) f\left(|Y|, s(Y)\right).$$

We mainly use Bit Error Rate(BER) to measure the error-recovery capability under a given SNR.

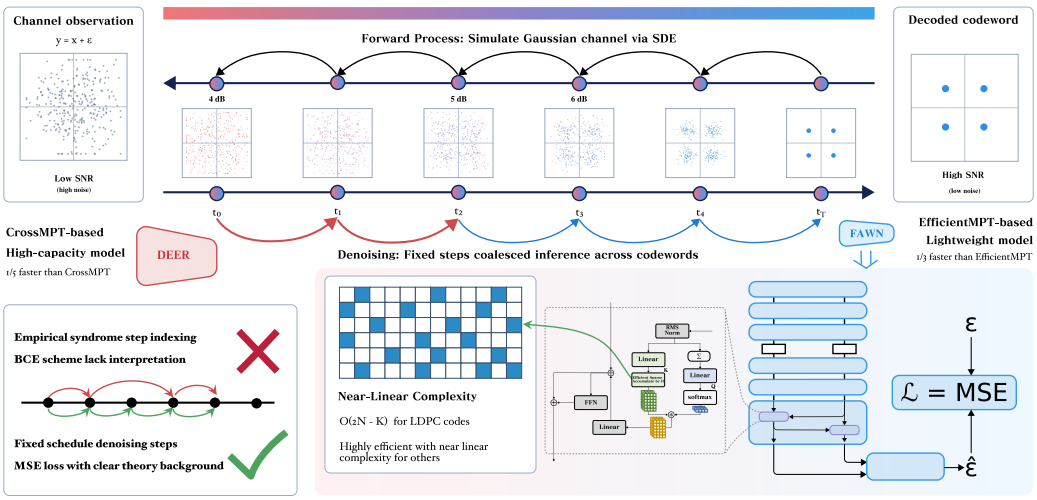


Figure 1: Pipeline design for our method. We incorporated a diffusion process and aligned different SNR values with diffusion time steps. Decoding is then modeled as the inverse process with $t_{\text{init}}$ determined based on the noise schedule and the corresponding SNR. Fixed-step inference and an optimized backbone model give rise to faster inference and better performance.

### 3.2 Diffusion-Based Neural Decoder Models

DDECC (Choukroun & Wolf, 2023) is an early work for diffusion-based decoders based on DDPM (Ho et al., 2020), while recently Zheng et al. (2026) use a simpler VE SDE model to facilitate flexible and efficient diffusion decoding. We here introduce the DDECC pipeline.

The DDECC is based on the forward diffusion

$$q(x_t \mid x_{t-1}) \sim \mathcal{N}(x_t; x_{t-1}, \beta_t \mathrm{I}).$$

So that

$$x_t = x_0 + \sqrt{\bar{\beta}}\varepsilon \sim \mathcal{N}(x_0, \bar{\beta}_t \mathrm{I}), \tag{1}$$

where $\bar{\beta}_t = \sum_{i=1}^{t} \beta_i$. Its update rule is then derived from the observation that

$$\mathbb{E}[x_{t-1} \mid x_t, x_0] = x_t - \frac{\beta_t}{\sqrt{\bar{\beta}_t}}\varepsilon \tag{2}$$

An estimator $\hat{\varepsilon}_\theta(x_t, t)$ is trained by minimizing $\mathcal{L}(\varepsilon, \hat{\varepsilon}_\theta(x_t, t))$, for estimating Eq. 2.

## 4 Rethinking Diffusion for Neural Decoding

### 4.1 A Closer Look at Diffusion-Based Neural Decoders

We continue from the DDECC setup to expose our argument. The step updates of DDECC rely on the approximation of $\varepsilon$ by $\hat{\varepsilon}_\theta(x_t, t)$. However, it is achieved implicitly by a neural network $f_\theta$ trained to minimize the binary cross-entropy loss with the multiplicative noise, namely $\underset{\theta}{\text{minimize}}\, \mathcal{L}_{\text{BCE}}(f_\theta(y, t), \text{sign}(x_0 \odot x_t))$.

DDECC then converts its binary multiplicative-noise prediction into a hard estimate of the transmitted codeword by $\hat{\varepsilon}_\theta = x_t - \text{sign}(x_t)\,\text{sign}(f(x_t, t))$, and its published reverse update is

$$x_{t-1} := x_t - \frac{\beta_t\sqrt{\bar{\beta}_t}}{\beta_t + \bar{\beta}_t}\left[x_t - \text{sign}(x_t)\,\text{sign}(f(x_t, t))\right] \tag{3}$$

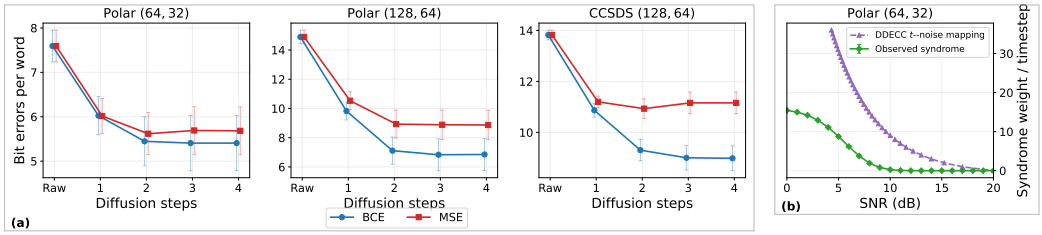


Figure 2: Average bit error estimation for failed codewords of three codes, between MSE- and BCE-trained DEER models, on the right: the observed syndrome for each noise level, and the actual noise level for DDECC diffusion time step, indexed by the syndrome $\gamma$

This rule is empirically effective, particularly when combined with the proposed syndrome-based line search. We note, however, that its connection to the theoretical framework needs further explanation. We intepreted it from an $x_0$-prediction perspective, since signal $x_0$ belongs to $S = \{\pm 1\}^N$, the term $\operatorname{sign}(x_t)\operatorname{sign}(f(x_t, t))$ is essentially an estimation of $x_0$ by elements in $S$, making it a constrained estimator of $x_0$, denoted by $\hat{x}_0$. Then $x_t - \hat{x}_0$ shall be an estimator of $\sqrt{\bar{\beta}}\varepsilon$, so instead of Eq. 3, $\varepsilon$ would be estimated as $\frac{1}{\sqrt{\bar{\beta}}}(x_t - \hat{x}_0)$, leading to an alternative update rule:

$$x_{t-1} := x_t - \frac{\beta_t}{\bar{\beta}_t}\left[x_t - \operatorname{sign}(x_t)\operatorname{sign}(f(x_t, t))\right] \tag{4}$$

The difference does not imply that the published DDECC rule is empirically invalid; rather, it shows that DDECC's practical update includes an additional scaling, approximately $\sqrt{\bar{\beta}_t}$, relative to the direct Gaussian-posterior plug-in update. We note two limitations for these two schedules: for one, the update formula only takes the sign of the prediction, while the amplitude representing prediction confidence was discarded during inference, causing an erroneous prediction to be harder for repair.

To show this, an experiment is conducted where our models are trained with MSE and BCE loss, respectively, with the BCE loss and inference schedule following DDECC. To show their trajectory, we select codewords for which both models fail after $4$ decoding iterations, their number of bit errors along their decoding trajectories is plotted in Fig. 2. In all cases, the MSE-trained model exhibits a lower-error trajectory than its BCE-trained counterpart, and they are more inclined to further optimize an already failed codeword, while the BCE-trained model can instead increase the number of bit errors in intermediate iterations.

The other limitation arises from the discrete projection inherent in $x_0$ prediction. The deviation above does not characterize how projecting $\hat{x}_0$ onto the discrete signal space $S$ alters the resulting diffusion dynamics. Our formulation instead directly regresses the continuous conditional velocity used during inference, avoiding the need to convert a projected binary prediction into a separately scaled continuous update.

We also notice that the time-scheduling mechanism introduces an additional mismatch. DDECC performs at most $N - K$ sequential decoding iterations, corresponding to the possible syndrome weights $\gamma \in \{0, \ldots, N - K\}$. Its forward process uses an additive variance schedule $\beta_t$, where the correspondence is determined by an empirical setup. At inference, DDECC computes the syndrome weight of the current observation,

$$\gamma = e(y) = |s(y)|_1 .$$

If $\gamma = 0$, the current hard decision is already a valid codeword and decoding can terminate. An update is otherwise performed, and the new diffusion state is accepted by setting $t = \gamma$.

As observed by Lei et al. (2025), the syndrome trajectory is non-smooth in the decoding trajectory. The actual noise level at time step $\gamma$ also has a large gap to the actual noise level that approaches the average syndrome count of $\gamma$, as the time steps' correspondence to the noise level is only empirically determined by $\beta$ without experimental guidance.

The DDECC-style inference is by design efficient, with average iterations approaching $1$, but in batched calculation it is rather infeasible to assign different computations for different codewords, making a fixed inference steps model more efficient than a model with low average steps.

Table 1: End-to-end decoding latency with a batch size of 2,048. All neural decoders use compiled FP32 inference. DDECC is evaluated using its official implementation and native $N - K$ iterations. The line-searched (LS) version is accelerated by our implementation, details are in Appendix E.

| **Decoder** | **Polar** $(64, 32)$ **Latency (ms)** | **BCH** $(63, 51)$ **Latency (ms)** |
|---|---|---|
| DDECC, native ($N - K$ iterations) | 2431 | 773 |
| DDECC, LS, 4 dB | 1780 | 799 |
| DDECC, LS, 5 dB | 962 | 786 |
| DDECC, LS, 6 dB | 168 | 403 |
| DEER, 1 step | 42 | 34 |
| FAWN L6, 1 step | **23** | **19** |
| Hybrid 1:1:2, 16 steps | 472 | 380 |

In Table 1, we compare our methods with a DDECC model. The implementation for adaptive Line Search is optimized with short-circuiting. Under the same setup (we refer to Appendix E), a fixed-step diffusion shall be more efficient than the uncoalesced adaptive one.

### 4.2 Channel Decoding as PF-ODE

We formulate our decoder as an application of stochastic interpolation and probability-flow dynamics (Albergo & Vanden-Eijnden, 2023; Lipman et al., 2023), with diffusion time directly parameterizing the channel noise level. Under the prescribed schedule, each $t$ corresponds to a fixed SNR. Hence, the channel SNR determines the initial time at inference without estimating $t$ from the received codeword.

Let $X_0$ be the transmitted signal and $Y$ an AWGN observation at the endpoint noise level. During training, we construct

$$X_t = X_0 + \eta_t(Y - X_0) + \sigma_t\varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, \mathrm{I}), \tag{5}$$

where $\eta_0 = 0$, $\eta_1 = 1$. Following the standard probability-flow formulation of such stochastic interpolations, the corresponding marginal velocity can be written as

$$u_t(x) = \dot{\eta}_t\mathbb{E}[Y - X_0 \mid X_t = x] + \dot{\sigma}_t\mathbb{E}[\varepsilon \mid X_t = x]. \tag{6}$$

We use aligned schedules $\sigma_t = c\eta_t$. Since

$$\frac{X_t - X_0}{\eta_t} = (Y - X_0) + c\varepsilon, \tag{7}$$

Eq. 6 reduces to

$$u_t(x) = \dot{\eta}_t\mathbb{E}\left[\frac{X_t - X_0}{\eta_t}\middle| X_t = x\right]. \tag{8}$$

This directly suggests regressing the normalized correction

$$V_t = \frac{X_t - X_0}{\eta_t} \tag{9}$$

with

$$\mathcal{L}_{\mathrm{MSE}} = \mathbb{E}\left[|v_\theta(X_t, t) - V_t|_2^2\right]. \tag{10}$$

The population minimizer of Eq. equation 10 is

$$v_\theta^\star(x, t) = \mathbb{E}[V_t \mid X_t = x, t], \tag{11}$$

so that $\dot{\eta}_t v_\theta^\star$ recovers the velocity in Eq. equation 8. Thus, rather than interpreting a BCE-trained decoding statistic as a continuous update, we directly regress the quantity used by the prescribed probability-flow dynamics.

At inference, the known or estimated channel SNR fixes the corresponding initial time $t_{\mathrm{init}}$, after which we integrate the learned dynamics toward $t = 0$ using a fixed schedule. The auxiliary perturbation $\sigma_t\varepsilon$ is used only in constructing training samples: inference starts directly from the received observation $Y$, without injecting additional noise.

### 4.3 Training and Inference

During training, for a batch, we continuously sample $t$ from $[t_{\min}, 1]$, and $v_\theta$ is trained to estimate $(X_t - X_0)/\eta_t$ with $X_t$ as input. This continuous sampling is shown in Zheng et al. (2026) to help boost performance.

During inference, given the noise level, $\sigma$, we determine $t_0$ so that the marginal distribution $\eta_{t_0}(Y - X_0) \sim \mathcal{N}(0, \sigma^2 \mathrm{I})$. For DEER, we omit the small auxiliary perturbation used during training, making inference slightly less noisy than the corresponding training path; FAWN uses $c = 0$ and therefore has no such mismatch. With a given step count $N$, the diffusion schedule $t_1, \cdots, t_N = 0$ is derived so that they are quadratically evenly spaced. Then the codeword is repeatedly updated using the derived rule

$$X_{t_{i+1}} - X_{t_i} = (\eta_{t_{i+1}} - \eta_{t_i}) v_\theta(X_{t_i}),$$

In practice, given the zero-codeword symmetry we use, the model is actually given by $v_\theta(X_t) = \mathrm{sign}(X_t) \cdot f_\theta(|X_t|, \text{syndrome}, \sigma_t)$.

With this training scheme, a crossMPT (Park et al., 2025)-backboned model, DEER is proposed. We follow other neural decoders' setup, assigning $L = 6$ layers and feature dimension $d = 128$.

Our framework is designed to run a fixed number of iterations, with noise levels being identical for every codeword in each step. Codewords that reach zero syndromes are still processed but no longer updated. Compared to Choukroun & Wolf (2023)'s empirical $n - k$ maximum inference steps, the inference steps are set to be 16 for low SNR regions of $4, 5$ dB, while the easier 6 dB region only requires $4$ iterations. Moreover, numerical results show that our models attain satisfactory performance when only $1$ step of diffusion is executed, on the harder $4$ dB noise range.

## 5 A Lightweight Implementation

### 5.1 Revisiting EfficientMPT

EfficientMPT was proposed by Park et al. (2026) to increase the efficiency of decoding transformers, which leads to an enormous reduction in memory and computation usage. Building on this design, we modified their implementations, where layers of ECC attention followed by residual FFNs were applied to iteratively extract information from syndromes to variables, and then from variables to syndromes. We proposed a modified implementation of ECC attention that is slightly faster and requires less computation, while being more stable. The details are in Appendix C.

In AECCT, a $1.58$-bit quantization scheme is introduced by Levy et al. (2024) that proposes ternary weights for linear layers. Although this design yields sparser weights and potentially reduces computation, an implementation on existing parallel computation hardware makes it difficult to show advantages. We nevertheless apply the pruning techniques to EfficientMPT and found that the feed-forward layer at the end of the transformer ends up with high sparsity, suggesting its potential for pruning. Therefore, its hidden dimension is halved without significant performance degradation.

These modifications have been successful in reducing complexity and latency by about $1/3$, but none of them has been toward performance optimizations. To this end, we experimented with adding $2$ more layers and doubling the attention heads. During these experiments, we encountered training instability and found the implementation glitch for layer norm, which we fixed in Appendix C. That fix enabled us to double the attention heads from $8$ to $16$, while also obtaining an $8$-layer variant that outperforms the $6$-layer variant, while the cost is still lower than the $6$-layer EfficientMPT. Since the overall computations do not change significantly with the change in the number of heads, we shall directly compare our $16$-head realization with others' $8$-head models.

This lighter model is denoted as FAWN. Following DEER, we set $d = 128$, while the number of layers $L$ was set to $6$ and $8$ for two variants: FAWN-L6 and FAWN-L8, the first matching EfficientMPT's parameters, and the second matching the amount of computation.

### 5.2 A Hybrid Method

For channels of different SNR, the complexity of decoding is closely related to the noise level: lower-magnitude noise causes fewer bits to flip and is easier to decode, while larger noise causing

multiple bits to flip might push the observed signal beyond the code's Maximum Likelihood decoding range. This brings the natural question of how much complexity and computation is required for the decoders at each noise level, which in our formulation corresponds to the stages of decoding: when the equivalent noise level decreases, more decoding iterations are applied.

From Table 4, it was found that our lighter model FAWN obtains a small advantage over the heavier DEER models, while at lower SNR regions, DEER wins over FAWN-L8 by a small margin. These results suggest that later, lower-noise stages may require less model capacity, suitable for the FAWN family models with their intermediate computational capacity, while higher noise requires more and might be better captured by the DEER models with their more expressive attention and heavier computations.

A heterogeneous diffusion design is thus brought forward to explore the diversity of those models, by assigning different stages of decoding to models that are suited to noise levels, while keeping the latency low. The design should be kept simple and independent of the specific linear codes involved. After experiments in Appendix B.2, a $1:1:2$ strategy is proposed.

To be specific, during our multi-step inference, the first $\frac{1}{4}$ of the stages are carried out by DEER to provide strong and reliable noise reduction, where the main directions for denoising are determined. Then a FAWN-L8 model takes over for another $\frac{1}{4}$ of the stages, where it continues the harder part with its highly sufficient architecture that captures most information. The remaining $\frac{1}{2}$ stages are left for the FAWN-L6 model to reconstruct the codeword in low-noise regimes. For example, in $16$-steps schedule, the first $4$ steps shall be completed by DEER, then followed by another $4$ steps by FAWN-L8, while the remaining $8$ steps are finished by FAWN-L6.

As shall be shown in Section 6.2, this design has an overall latency comparable to FAWN-L8, while the performance exceeds both DEER and the FAWN models, suggesting this hybrid diffusion scheme achieves the best tradeoff of latency versus the decoding capability, and places each model in its better-suited noise regions.

# 6 Experiments

In this section, we present the model's setup, training, inference, and result comparisons, along with the computation and latency measurements.

## 6.1 Model Setup

Our heavier model, DEER, is based on the crossMPT backbone with our diffusion model setup, while our lighter FAWN model is based on the EfficientMPT backbone. DEER and FAWN-L6 use $6$ layers, while FAWN-L8 uses $8$ layers. FAWN-L6 and FAWN-L8 share a doubled attention head count of $16$, DEER uses the default $8$-head setup. They all use $128$ internal dimensions.

The reported BERs are calculated using a simulation that adaptively simulates codewords and corrects errors. The stochastic measurement inevitably introduces uncertainties, for which we shall provide confidence intervals in our provided source code, and our comparison is mainly focused on $4$ dB, where more errors are visible and thus the results are more reliable.

More details for training, evaluation, and benchmarking are provided in Appendix D.

## 6.2 Comparison

We compare $-\ln \text{BER}$ for some methods in Table 2, where the complete results are in Table 5 and Table 4. Some cases are presented in Fig. 3.

These results show improvements in the $-\ln \text{BER}$ of our models over others, especially at the $4$ dB noise level, with often a $0.5$ increase in $-\ln \text{BER}$, corresponding to around a $39\%$ reduction in bit errors. Since $4$ dB has higher noise, and the BER estimations are more reliable for low SNR noise, this increase suggests solid improvements. In other cases, DEER also shows very competitive performance, with FAWN-L8 catching up at Polar $(64, 32)$, BCH $(63, 36)$, BCH $(63, 45)$, and LDPC $(121, 60)$, CCSDS $(128, 64)$.

Table 2: $-\ln \mathrm{BER}$ comparison across decoding methods. Higher values are better. **Best values** are bolded, while the second best values are underlined. ‘ Multi-step method are marked by $^+$. ‘Classical’ refers to SCL L=8 for Polar codes, OSD t=2 for BCH codes, and SPA with 50 iterations for LDPC codes

| Method | SNR | Polar | | | | | BCH | | | | LDPC | | | | CCSDS | MacKay |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | (64, 32) | (64, 48) | (128, 64) | (128, 86) | (128, 96) | (31, 16) | (63, 36) | (63, 45) | (63, 51) | (49, 24) | (121, 60) | (121, 70) | (121, 80) | (128, 64) | (96, 48) |
| DDECC$^+$ | 4 dB | 6.93 | 5.96 | 9.11 | 7.60 | 7.16 | – | 5.11 | 5.61 | 5.26 | 6.32 | 5.38 | 6.79 | 7.59 | 7.81 | 8.12 |
| | 5 dB | 9.51 | 8.04 | 12.90 | 10.81 | 10.30 | – | 7.09 | 7.94 | 7.40 | 9.01 | 8.73 | 11.13 | 12.17 | 12.48 | 11.88 |
| | 6 dB | 12.79 | 10.98 | 16.30 | 15.17 | 13.19 | – | 9.82 | 11.36 | 10.49 | 12.36 | 14.17 | 16.93 | 16.89 | 17.66 | 15.93 |
| TECCT$^+$ | 4 dB | 7.46 | 6.41 | – | – | – | – | 5.70 | 6.77 | 5.97 | – | 5.55 | 6.83 | 7.94 | – | – |
| | 5 dB | 10.16 | 8.40 | – | – | – | – | 7.97 | 9.78 | 8.49 | – | 9.11 | 11.13 | 12.26 | – | – |
| | 6 dB | 13.53 | 11.07 | – | – | – | – | 10.80 | 13.62 | 12.00 | – | 14.57 | 17.21 | 18.17 | – | – |
| EfficientMPT | 4 dB | 7.44 | 6.36 | 7.05 | – | – | 7.01 | 4.91 | 5.73 | – | – | 5.62 | 6.96 | 7.95 | – | – |
| | 5 dB | 10.10 | 8.58 | 10.69 | – | – | 9.38 | 6.64 | 8.05 | – | – | 9.04 | 11.31 | 12.40 | – | – |
| | 6 dB | 13.37 | 11.02 | 14.94 | – | – | 12.54 | 9.23 | 11.11 | – | – | 14.73 | 16.97 | 17.99 | – | – |
| ECCT | 4 dB | 6.99 | 6.36 | 5.92 | 6.31 | 6.31 | 6.39 | 4.86 | 5.60 | 5.66 | 6.68 | 5.17 | 6.40 | 7.41 | 6.88 | 7.38 |
| | 5 dB | 9.44 | 8.46 | 8.64 | 9.01 | 9.12 | 8.29 | 6.65 | 7.79 | 7.89 | 9.53 | 8.31 | 10.21 | 11.51 | 10.90 | 10.72 |
| | 6 dB | 12.32 | 11.09 | 12.18 | 12.45 | 12.47 | 10.66 | 9.10 | 10.93 | 11.01 | 13.30 | 13.30 | 16.11 | 16.44 | 15.90 | 14.83 |
| CrossMPT | 4 dB | 7.50 | 6.51 | 7.52 | 7.51 | 7.15 | 6.98 | 5.03 | 5.90 | 5.78 | 6.68 | 5.74 | 7.06 | 7.99 | 8.04 | 7.97 |
| | 5 dB | 9.97 | **8.70** | 11.21 | 10.83 | 10.15 | 9.25 | 6.91 | 8.20 | 8.08 | 9.52 | 9.26 | 11.39 | 12.75 | 11.88 | 11.77 |
| | 6 dB | 13.31 | 11.31 | 14.76 | 15.24 | 13.13 | 12.48 | 9.37 | 11.62 | 11.41 | 13.19 | 14.78 | 17.52 | 18.15 | 17.50 | 15.52 |
| Hybrid$^+$ | 4 dB | **7.91** | **6.62** | **10.75** | **8.42** | **7.77** | 7.49 | 6.04 | 6.71 | 6.33 | **7.81** | **6.45** | **8.03** | **8.85** | **8.78** | **8.98** |
| | 5 dB | **10.48** | 8.68 | **13.98** | **11.89** | **10.42** | 9.97 | 8.44 | 9.78 | **9.01** | **11.00** | **10.32** | **13.02** | **14.03** | **13.80** | **12.50** |
| | 6 dB | **13.94** | **11.33** | 17.44 | **16.04** | **13.45** | 13.08 | 11.46 | 13.52 | 12.67 | **14.77** | **15.04** | **18.61** | **18.43** | **18.08** | **16.42** |
| DEER 1-step | 4 dB | 7.82 | **6.62** | 9.51 | 7.74 | 7.41 | 7.26 | 5.54 | 6.20 | 6.13 | 7.33 | 6.07 | 7.50 | 8.33 | 7.88 | 8.34 |
| | 5 dB | 10.25 | 8.63 | 13.30 | 10.93 | 10.41 | 9.82 | 7.64 | 8.73 | 8.73 | 10.23 | 9.78 | 11.91 | 13.31 | 12.62 | 11.93 |
| | 6 dB | 13.77 | 11.17 | 16.10 | 14.41 | 13.21 | 12.88 | 10.68 | 12.08 | 12.50 | 14.05 | 12.78 | 16.80 | 16.72 | 17.08 | 15.43 |
| Classical | 4 dB | 6.99 | 5.05 | 8.42 | 5.67 | 5.32 | **7.59** | **10.28** | **8.26** | **6.77** | 6.12 | 5.71 | 7.07 | 7.89 | 8.04 | 8.02 |
| | 5 dB | 9.91 | 7.26 | 12.79 | 9.43 | 8.72 | **9.98** | **14.73** | **11.89** | **9.67** | 8.46 | 8.75 | 10.55 | 11.89 | 12.16 | 11.21 |
| | 6 dB | 13.69 | 10.22 | **18.33** | 15.38 | 12.37 | **13.07** | **>20.20** | **17.22** | **13.52** | 11.97 | 13.24 | 15.94 | 16.85 | 17.75 | 14.76 |

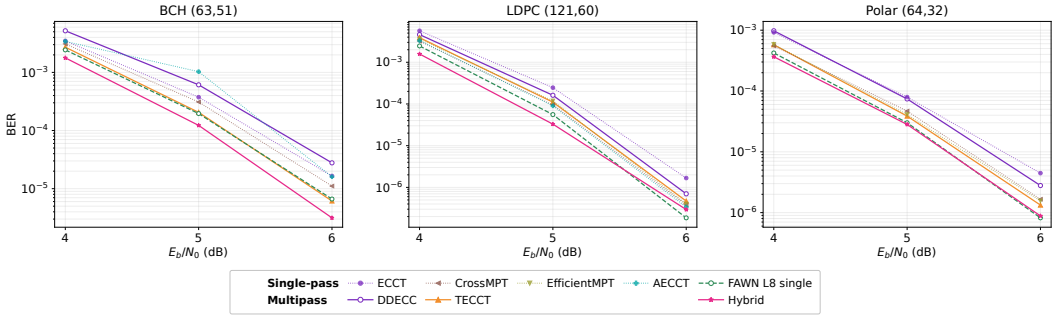


Figure 3: BER curves of different methods across different linear codes

In high-SNR regions, the simulation noise makes estimations less accurate. However, we note that except for a few exceptions when DEER shows degraded performance at 6 dB, DEER remains competitive, with FAWN-L8 outperforming DEER especially at high SNR, and FAWN-L6 being slightly behind L8 variants across different code families. The hybrid method receives consistently higher $-\ln \mathrm{BER}$ across all families, securing the best or second best in most cases.

The single-step comparison in Table 5 once again exhibits this trend, with DEER leading in the 4 dB region, slightly weaker at 5 dB, and degrading in some cases at 6 dB. FAWN-L8 is moderately well, and L6 falls behind, but FAWN-L6 still achieves lower BER against previous classic 1-step decoders.

### 6.3 Accuracy-Efficiency Tradeoff

In Table 3, we compare latencies for the involved methods measured on our hardware with identical settings. We set the same precisions and use torch.compile for optimal speed. Batch size was set to 2048 for the three shorter codes, while the last Tanner $(310, 128)$ code makes ECCT out of memory and hence a smaller batch size is set. Setup for latency measurement is provided in Appendix D.3.

Compared to their backbones, DEER gains around $1/5$ latency for its efficient implementation, and FAWN's optimizations have reduced around $1/3$ latency across various code families. FAWN-L8 has attained higher decoding performance under the same latency and computation budgets as 6-layer EfficientMPT.

Table 3: Inference Latency across different codes and model configurations. The latency unit is milliseconds. 'Classical' refers to SCL L=8 for Polar codes, OSD t=2 for BCH codes, and SPA with 50 iterations for LDPC codes.

| | Single-step | | | | | | | 4-steps | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Code | Classical | FAWN L6 | FAWN L8 | EfficientMPT | DEER | CrossMPT | ECCT | FAWN L6 | FAWN L8 | Hybrid | DEER | Batch |
| Polar $(128, 64)$ | 20.29 | 43.5 | 57.5 | 81.5 | 75.2 | 106.9 | 139.3 | 174.9 | 176.6 | 177.9 | 226.1 | 2048 |
| BCH $(63, 45)$ | 4.59 | 20.8 | 27.2 | 33.2 | 38.1 | 45.3 | 53.8 | 82.9 | 109.5 | 106.3 | 151.9 | 2048 |
| CCSDS $(128, 64)$ | 8.50 | 43.5 | 57.7 | 81.1 | 74.4 | 106.9 | 139.1 | 174.2 | 173.1 | 177.5 | 221.6 | 2048 |
| Tanner $(310, 128)$ | 3.51 | 16.8 | 22.2 | 38.7 | 36.4 | 55.8 | 83.8 | 66.1 | 87.6 | 91.6 | 145.1 | 256 |

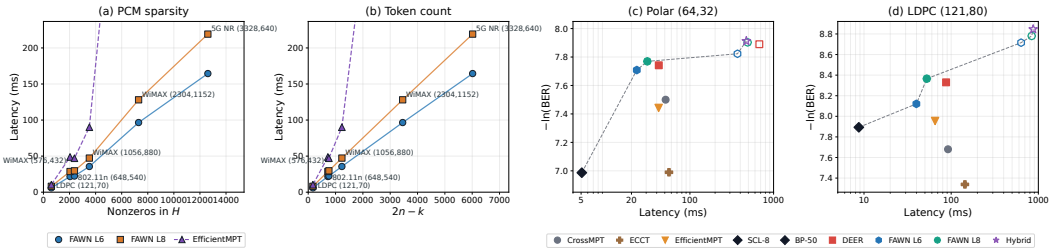


Figure 4: Latency versus code parameters. We measure latencies for a series of long LDPC code, demonstrating the linear scaling of run-time with code parameters.

FAWN's latency grows linearly for LDPC codes with the total input length $2N - K$, and also the number of nonzero elements in check matrix $H$, denoted as $h$, as shown from the tests in Fig.4. This clear linear scale for the parameter $2N - K$ follows from our modified ECC Attention implementation (Appendix C). Rather than using dense matrix multiplication, a sparse kernel processes only the nonzero entries in $H$, yielding $O(h)$ complexity. This computation is dominated in LDPC codes by the sequence handling of our model, which takes $O(2N - K)$ complexity. The general overall complexity for any code family shall be $O(2N - K + h)$.

Latency comparison against other diffusion models is more intricate, as their measurements were mostly based on single-step diffusion latency and the average diffusion steps, with latency estimated by the product of these two values. In practice, however, the real wall clock time might differ depending on the specific implementations, largely due to the batch-parallel nature of the GPU and the overhead computation caused by forcing a batch of codewords to go through the same amount of computations. We include a comparison of our inference latency with DDECC in Appendix E

We address a more general comparison in two points: Our model's 1-step diffusion results are already on par with the best existing diffusion models, with lower computational cost. And for actual runtime, shown in Appendix E, the inference speed exceeds the optimized DDECC implementation at 4 and 5 dB while being on par with it for 6 dB.

As shown in Figure 4, our model improves the observed latency-performance frontier, with the single-step being competitive in both performance and speed among the evaluated neural decoders, while the diffusion models reach the strongest measured decoding performance, but at significantly higher latency.

## 7 Conclusion

In this work, a simpler framework for diffusion-based neural decoders is discussed, mitigating several inconsistencies in the neural decoder design. By adopting the new framework, a diffusion-based decoder model DEER is proposed and achieves competitive BER among different families of codes. Following this success, an EfficientMPT-based variant FAWN was proposed with lower computational costs and slightly degraded performance compared to DEER. This gap is narrowed when we convert the computational savings into higher capacity of the model, through introducing 2 extra layers for FAWN-L8. By heterogeneously passing different decoding stages to different decoders, we were able to achieve faster decoding with even better performance.

### AI USE STATEMENT

In this work, we used generative AI tools to aid the development of the program, to execute the experiments, to rephrase the writing of the paper, and to aid of drawing part of the plots and figures. We have not used generative AI tools for generation of the research idea, nor for initial writing of the paper. We have reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

### REPRODUCIBILITY STATEMENT

We shall provide all training and evaluation source code related to this work through GitHub once this paper is accepted. To facilitate reproducing our results, all the weights and results for our models, mentioned in this paper, shall be open for download. This work requires no dataset except for AWGN simulation pairs, which are generated in real time by the program.

## REFERENCES

Michael Samuel Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. In **The Eleventh International Conference on Learning Representations**, 2023. URL https://arxiv.org/abs/2209.15571.

Amir Bennatan, Yoni Choukroun, and Pavel Kisilev. Deep Learning for Decoding of Linear Codes - A Syndrome-Based Approach. In **2018 IEEE International Symposium on Information Theory (ISIT)**, pp. 1595–1599, June 2018. doi: 10.1109/ISIT.2018.8437530. URL https://ieeexplore.ieee.org/document/8437530. ISSN: 2157-8117.

Sanghyeon Cho, Taewoo Park, Seong-Joon Park, Dae-Young Yun, Hee-Youl Kwak, Sang-Hyo Kim, and Yongjune Kim. Spectral-Aligned Pruning for Universal Error-Correcting Code Transformers, February 2026. URL https://arxiv.org/abs/2602.01602v3.

Yoni Choukroun and Lior Wolf. Error Correction Code Transformer. In **Advances in Neural Information Processing Systems**, volume 35, pp. 38695–38705. Curran Associates, Inc., 2022. doi: 10.52202/068431-2804. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/fcd3909db30887ce1da519c4468db668-Abstract-Conference.html.

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.

M.P.C. Fossorier and Shu Lin. Soft-decision decoding of linear block codes based on ordered statistics. **IEEE Transactions on Information Theory**, 41(5):1379–1396, 1995. doi: 10.1109/18.412683.

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, March 2017. doi: 10.1109/CISS.2017.7926071. URL https://ieeexplore.ieee.org/document/7926071.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), **Advances in Neural Information Processing Systems**, volume 33, pp. 6840–6851. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf.

Sanxin Jiang, Jiro Katto, and Heming Sun. Rddm: A rate-distortion guided diffusion model for learned image compression enhancement. **IEEE Journal on Emerging and Selected Topics in Circuits and Systems**, 15(2):186–199, 2025. doi: 10.1109/JETCAS.2025.3563228.

F.R. Kschischang, B.J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. **IEEE Transactions on Information Theory**, 47(2):498–519, 2001. doi: 10.1109/18.910572.

Haoyu Lei, Chin Wa Lau, Kaiwen Zhou, Nian Guo, and Farzan Farnia. Syndrome-Flow Consistency Model Achieves One-step Denoising Error Correction Codes, December 2025. URL https://arxiv.org/abs/2512.01389v2.

Matan Levy, Yoni Choukroun, and Lior Wolf. Accelerating Error Correction Code Transformers, October 2024. URL https://arxiv.org/abs/2410.05911v1.

Jinyuan Li, Shuai Xiao, and Weijun Fang. ProductMPT: Message-Passing Transformer for Product Codes. In **2026 IEEE International Symposium on Information Theory (ISIT)**, pp. 1–6, June 2026. doi: 10.1109/ISIT62367.2026.11653848. URL https://ieeexplore.ieee.org/abstract/document/11653848. ISSN: 2157-8117.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In **The Eleventh International Conference on Learning Representations**, 2023. URL https://openreview.net/forum?id=PqvMRDCJT9t.

Guan-Horng Liu, Arash Vahdat, De-An Huang, Evangelos A Theodorou, Weili Nie, and Anima Anandkumar. $I^2$sb: Image-to-image schrödinger bridge. **arXiv preprint arXiv:2302.05872**, 2023.

Loren Lugosch and Warren J. Gross. Neural offset min-sum decoding. In **2017 IEEE International Symposium on Information Theory (ISIT)**, pp. 1361–1365, June 2017. doi: 10.1109/ISIT.2017.8006751. URL https://ieeexplore.ieee.org/document/8006751. ISSN: 2157-8117.

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, September 2016. doi: 10.1109/ALLERTON.2016.7852251. URL https://ieeexplore.ieee.org/document/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, February 2018. ISSN 1941-0484. doi: 10.1109/JSTSP.2017.2788405. URL https://ieeexplore.ieee.org/document/8242643.

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 **International Conference on Learning Representations**, volume 2025, pp. 56710–56728, May 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/8f0942c43fcfba4cc66a859b9fcb1bba-Abstract-Conference.html.

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=gFvRRCnQvX.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising Diffusion Implicit Models. In **International Conference on Learning Representations**, 2021a. URL https://mlanthology.org/iclr/2021/song2021iclr-denoising/.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In **International Conference on Learning Representations**, 2021b. URL https://openreview.net/forum?id=PxTIG12RRHS.

Ido Tal and Alexander Vardy. List decoding of polar codes. **IEEE Transactions on Information Theory**, 61(5):2213–2226, 2015. doi: 10.1109/TIT.2015.2410251.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), **Advances in Neural Information Processing Systems**, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Zongsheng Yue, Jianyi Wang, and Chen Change Loy. Resshift: Efficient diffusion model for image super-resolution by residual shifting. In **Thirty-seventh Conference on Neural Information Processing Systems**, 2023. URL https://openreview.net/forum?id=ZIyAHaLlsn.

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, July 2024. doi: 10.1109/ISIT57864.2024.10619193. URL https://ieeexplore.ieee.org/document/10619193. ISSN: 2157-8117.

Jian Zheng, Jiahuan Ni, Zongyao Li, Yutai Sun, Xiaosi Tan, Yongming Huang, and Chuan Zhang. TCD: Time-Conditioned Diffusion Decoding for Binary Short Block Codes. **IEEE Transactions on Cognitive Communications and Networking**, 12:11966–11980, 2026. ISSN 2332-7731. doi: 10.1109/TCCN.2026.3712754. URL https://ieeexplore.ieee.org/abstract/document/11674355.

# A Full Result Table

We present the full result table here, which compares our methods with other methods: for Table 4, the diffusion models that require multiple decoding steps; and for Table 5, methods that achieve decoding in 1 step. DEER refers to our heavier CrossMPT (Park et al., 2025)-backbone model, and FAWN refers to our lighter EfficientMPT(Park et al., 2026)-based model. Hybrid refers to the DEER and FAWN combined decoding methods.

Throughout the comparison, except FAWN-L8, which involves $L = 8$ layers, all other methods have $L = 6$ layers. As far as we know, all methods have a feature dimension $d = 128$ and 8 heads, except for FAWN families with 16 heads.

Though it might not be strictly fair to compare others with FAWN-L6 and FAWN-L8 since they have more heads and L8 has more layers, we note that FAWN-L8 has decoding latency and computation at a lower level than EfficientMPT at 6 layers, and head counts do not increase the computation, slightly justifying the comparison.

We also added classical decoders into Table 4, whose details can be found in Appendix D.4. We use, for Polar codes, an SCL algorithm with $l = 8$, OSD with $t = 2$ for BCH codes, and the SPA algorithm with 50 iterations for LDPC codes.

We acknowledge that though the current neural decoders outperform SCL and SPA algorithms, they were not competitive for BCH codes, with an extremely large performance gap for BCH $(63, 36)$. Classical decoders are around $2 - 6$ times faster than our fastest one-step neural decoders, as shown in Table 3 Given that the OSD algorithm has an approximately $O(N^t)$ complexity for fixed-rate code, we find a potential need to pursue the performance of neural decoders for longer BCH code.

# B Ablation Study

## B.1 Ablation on DEER model

An ablation study is carried out to investigate the effect of the model's setups for our introduced modifications besides the diffusion schemes. As shown in Table 6, we made the following observations

1. The additional noise formulation has not demonstrated significant impact on BER. Thus, it is removed for FAWN.
2. We found that noise (also an equivalent time embedding) embeddings do not have a significant impact, while the position-wise embeddings are essential to the stability of the model. Changing to shared embedding types collapsed the model for Polar $(128, 64)$ and CCSDS $(128, 64)$. Instead of the current multiplicative embeddings, additive and MLP-style embeddings are being tested, and no significant changes were observed.
3. We found that for low SNR, a setup with more inference steps exhibits higher computational capacity, while for higher SNR, fewer steps are sufficient to denoise efficiently, with more steps being harmful for performances. We therefore empirically set our default step count by setting 16 steps for 4 and 5 dB, and 4 steps for 6 dB.
4. The inference schedule defines how the timesteps are determined during the decoding process, though a square schedule was empirically decided that allocates more steps at the higher-noise end. A linear schedule inference decoder demonstrated error-correction ability just as well.
5. It is of vital importance to investigate if alternating the form of the check matrix could perturb the results, as the defined form might possess more patterns and structures, while a systematic form might be sparser in certain regions. Our findings suggest the performance might be unrelated to the PCM's form.
6. Training schedule affects how the training data are sampled, and compared to linear sampling of $\eta$, our default sampling on $t$, by uniform on $\eta^2$, allocates sufficiently more data samples at harder low SNR regions, while still having enough data samples for high SNR region performances. The linear-uniform sampling in $\eta$ shows earlier convergence and degraded/collapsed performance.

Table 4: $-\ln$ BER Comparison with diffusion-based decoding methods. Higher values are better. **Best values** are bolded, while the second best values are underlined.

| Code | $(n, k)$ | SNR | DDECC | TECCT | DEER | FAWN-L6 | FAWN-L8 | Hybrid | Classical |
|---|---|---|---|---|---|---|---|---|---|
| Polar | $(64, 32)$ | 4 dB | 6.93 | 7.46 | 7.89 | 7.82 | 7.90 | **7.91** | 6.99 |
| | | 5 dB | 9.51 | 10.16 | 10.45 | **10.56** | 10.41 | 10.48 | 9.91 |
| | | 6 dB | 12.79 | 13.53 | 13.77 | 13.86 | 13.92 | **13.94** | 13.69 |
| | $(64, 48)$ | 4 dB | 5.96 | 6.41 | **6.67** | 6.58 | 6.55 | 6.62 | 5.05 |
| | | 5 dB | 8.04 | 8.40 | 8.64 | 8.67 | **8.71** | 8.68 | 7.26 |
| | | 6 dB | 10.98 | 11.07 | 11.26 | 11.14 | 11.17 | **11.33** | 10.22 |
| | $(128, 64)$ | 4 dB | 9.11 | – | 10.44 | 8.99 | 9.40 | **10.75** | 8.42 |
| | | 5 dB | 12.90 | – | 13.63 | 12.85 | 13.46 | **13.98** | 12.79 |
| | | 6 dB | 16.30 | – | 16.15 | 16.96 | 16.68 | 17.44 | **18.33** |
| | $(128, 86)$ | 4 dB | 7.60 | – | **8.43** | 8.07 | 8.04 | 8.42 | 5.67 |
| | | 5 dB | 10.81 | – | **11.93** | 11.47 | 11.77 | 11.89 | 9.43 |
| | | 6 dB | 15.17 | – | 15.59 | 15.45 | 15.81 | **16.04** | 15.38 |
| | $(128, 96)$ | 4 dB | 7.16 | – | 7.73 | 6.94 | 6.82 | **7.77** | 5.32 |
| | | 5 dB | 10.30 | – | **10.54** | 9.96 | 9.89 | 10.42 | 8.72 |
| | | 6 dB | 13.19 | – | 13.01 | 13.06 | 13.17 | **13.45** | 12.37 |
| BCH | $(31, 16)$ | 4 dB | – | – | 7.52 | 7.33 | 7.34 | 7.49 | **7.59** |
| | | 5 dB | – | – | **9.99** | 9.80 | 9.88 | 9.97 | 9.98 |
| | | 6 dB | – | – | 13.01 | 12.82 | 12.93 | **13.08** | 13.07 |
| | $(63, 36)$ | 4 dB | 5.11 | 5.70 | 5.86 | 5.76 | 5.92 | 6.04 | **10.28** |
| | | 5 dB | 7.09 | 7.97 | 8.12 | 7.94 | 8.16 | 8.44 | **14.73** |
| | | 6 dB | 9.82 | 10.80 | 10.90 | 10.82 | 10.97 | 11.46 | **>20.20** |
| | $(63, 45)$ | 4 dB | 5.61 | 6.77 | 6.57 | 6.50 | 6.57 | 6.71 | **8.26** |
| | | 5 dB | 7.94 | 9.78 | 9.26 | 9.33 | 9.17 | 9.60 | **11.89** |
| | | 6 dB | 11.36 | 13.62 | 12.86 | 12.85 | 12.78 | 13.52 | **17.22** |
| | $(63, 51)$ | 4 dB | 5.26 | 5.97 | 6.34 | 6.17 | 6.20 | 6.33 | **6.77** |
| | | 5 dB | 7.40 | 8.49 | 8.93 | 8.95 | 8.77 | 9.01 | **9.67** |
| | | 6 dB | 10.49 | 12.00 | 12.36 | 12.22 | 12.41 | 12.67 | **13.52** |
| LDPC | $(49, 24)$ | 4 dB | 6.32 | – | 7.69 | 7.35 | 7.45 | **7.81** | 6.12 |
| | | 5 dB | 9.01 | – | 10.81 | 10.44 | 10.42 | **11.00** | 8.46 |
| | | 6 dB | 12.36 | – | 14.06 | 14.53 | 14.43 | **14.77** | 11.97 |
| | $(121, 60)$ | 4 dB | 5.38 | 5.55 | 6.24 | 6.26 | 6.32 | **6.45** | 5.71 |
| | | 5 dB | 8.73 | 9.11 | 9.58 | 10.22 | **10.39** | 10.32 | 8.75 |
| | | 6 dB | 14.17 | 14.57 | 13.34 | **16.61** | 16.09 | 15.04 | 13.24 |
| | $(121, 70)$ | 4 dB | 6.79 | 6.83 | 7.93 | 7.75 | 7.74 | **8.03** | 7.07 |
| | | 5 dB | 11.13 | 11.13 | 12.58 | 12.31 | 12.64 | **13.02** | 10.55 |
| | | 6 dB | 16.93 | 17.21 | 16.85 | 18.21 | 17.92 | **18.61** | 15.94 |
| | $(121, 80)$ | 4 dB | 7.59 | 7.94 | 8.79 | 8.72 | 8.48 | **8.85** | 7.89 |
| | | 5 dB | 12.17 | 12.26 | 13.83 | 13.62 | 13.89 | **14.03** | 11.89 |
| | | 6 dB | 16.89 | 18.17 | 17.48 | **18.83** | 18.61 | 18.43 | 16.85 |
| CCSDS | $(128, 64)$ | 4 dB | 7.81 | – | 8.39 | 8.55 | 8.57 | **8.78** | 8.04 |
| | | 5 dB | 12.48 | – | 13.58 | 13.43 | 13.42 | **13.80** | 12.16 |
| | | 6 dB | 17.66 | – | 18.14 | **18.20** | 18.08 | 18.08 | 17.75 |
| MacKay | $(96, 48)$ | 4 dB | 8.12 | – | 8.98 | 8.98 | **9.03** | 8.98 | 8.02 |
| | | 5 dB | 11.88 | – | 12.50 | 12.75 | **12.96** | 12.50 | 11.21 |
| | | 6 dB | 15.93 | – | **16.42** | 16.24 | 16.03 | **16.42** | 14.76 |

7. Our training was in BF16, and with our implementation optimizations, we were able to process more samples per batch. The ablations showed slightly downgraded performance for 128 as batch size, while slightly improved for a batch size of 1024.

8. Since our main evaluation SNR range is $4 - 6$ dB, it raises the question of whether harder cases during training help with the decoding, or smaller portions of testing SNR range samples during training make the model learn less about decoding in those regions. From the ablation results, a narrower training sample SNR range potentially helps the performance, achieving improvements in 9 out of 12 evaluated entries.

Table 5: $-\ln$ BER Comparison with single step decoding methods. Higher values are better. **Best values** are bolded, while the second best values are underlined.

| Code | $(n, k)$ | SNR | CrossMPT | ECCT | EfficientMPT | AECCT | DEER | FAWN-L6 | FAWN-L8 |
|---|---|---|---|---|---|---|---|---|---|
| Polar | $(64, 32)$ | 4 dB | 7.50 | 6.99 | 7.44 | – | **7.82** | 7.71 | 7.77 |
| | | 5 dB | 9.97 | 9.44 | 10.10 | – | 10.25 | **10.56** | 10.41 |
| | | 6 dB | 13.31 | 12.32 | 13.37 | – | 13.77 | 13.81 | **14.01** |
| | $(64, 48)$ | 4 dB | 6.51 | 6.36 | 6.36 | 6.54 | **6.62** | 6.50 | 6.58 |
| | | 5 dB | 8.70 | 8.46 | 8.58 | 8.51 | 8.63 | **8.76** | 8.54 |
| | | 6 dB | 11.31 | 11.09 | 11.02 | 11.12 | 11.17 | **11.33** | 10.93 |
| | $(128, 64)$ | 4 dB | 7.52 | 5.92 | 7.05 | – | **9.51** | 7.70 | 8.12 |
| | | 5 dB | 11.21 | 8.64 | 10.69 | – | **13.30** | 11.82 | 12.36 |
| | | 6 dB | 14.76 | 12.18 | 14.94 | – | 16.10 | 16.24 | **17.23** |
| | $(128, 86)$ | 4 dB | 7.51 | 6.31 | – | 7.28 | **7.74** | 7.29 | 7.32 |
| | | 5 dB | 10.83 | 9.01 | – | 10.60 | **10.93** | 10.51 | 10.69 |
| | | 6 dB | **15.24** | 12.45 | – | 14.59 | 14.41 | 14.93 | 15.00 |
| | $(128, 96)$ | 4 dB | 7.15 | 6.31 | – | 6.79 | **7.41** | 6.94 | 6.82 |
| | | 5 dB | 10.15 | 9.12 | – | 9.68 | 10.41 | 9.96 | 9.89 |
| | | 6 dB | 13.13 | 12.47 | – | 12.93 | **13.21** | 13.06 | 13.17 |
| BCH | $(31, 16)$ | 4 dB | 6.98 | 6.39 | 7.01 | 7.21 | **7.26** | 7.16 | 7.15 |
| | | 5 dB | 9.25 | 8.29 | 9.38 | 9.47 | **9.82** | 9.62 | 9.78 |
| | | 6 dB | 12.48 | 10.66 | 12.54 | 12.45 | **12.88** | 12.82 | 12.81 |
| | $(63, 36)$ | 4 dB | 5.03 | 4.86 | 4.91 | 4.90 | **5.54** | 5.35 | 5.41 |
| | | 5 dB | 6.91 | 6.65 | 6.64 | 6.64 | **7.64** | 7.32 | 7.53 |
| | | 6 dB | 9.37 | 9.10 | 9.23 | 9.19 | **10.68** | 10.24 | 10.14 |
| | $(63, 45)$ | 4 dB | 5.90 | 5.60 | 5.73 | 5.83 | **6.20** | 6.14 | 6.17 |
| | | 5 dB | 8.20 | 7.79 | 8.05 | 8.15 | **8.73** | 8.58 | 8.60 |
| | | 6 dB | 11.62 | 10.93 | 11.11 | 11.52 | 12.08 | **12.41** | 12.19 |
| | $(63, 51)$ | 4 dB | 5.78 | 5.66 | – | 5.68 | **6.13** | 5.95 | 6.02 |
| | | 5 dB | 8.08 | 7.89 | – | 7.88 | **8.73** | 8.47 | 8.53 |
| | | 6 dB | 11.41 | 11.01 | – | 11.04 | **12.50** | 11.57 | 11.92 |
| LDPC | $(49, 24)$ | 4 dB | 6.68 | 6.68 | – | 6.67 | **7.33** | 6.85 | 7.00 |
| | | 5 dB | 9.52 | 9.53 | – | 9.35 | **10.23** | 9.82 | 9.92 |
| | | 6 dB | 13.19 | 13.30 | – | 13.56 | **14.05** | 13.55 | 13.94 |
| | $(121, 60)$ | 4 dB | 5.74 | 5.17 | 5.62 | 5.71 | **6.07** | 5.89 | 6.01 |
| | | 5 dB | 9.26 | 8.31 | 9.04 | 9.31 | 9.78 | 9.66 | **9.80** |
| | | 6 dB | 14.78 | 13.30 | 14.73 | 14.90 | 12.78 | 15.28 | **15.50** |
| | $(121, 70)$ | 4 dB | 7.06 | 6.40 | 6.96 | 7.05 | **7.50** | 7.22 | 7.46 |
| | | 5 dB | 11.39 | 10.21 | 11.31 | 11.40 | 11.91 | 11.71 | **12.03** |
| | | 6 dB | **17.52** | 16.11 | 16.97 | 17.30 | 16.80 | 17.11 | 16.94 |
| | $(121, 80)$ | 4 dB | 7.99 | 7.41 | 7.95 | 7.98 | 8.33 | 8.12 | **8.37** |
| | | 5 dB | 12.75 | 11.51 | 12.40 | 12.60 | **13.31** | 12.69 | 13.29 |
| | | 6 dB | 18.15 | 16.44 | 17.99 | 18.10 | 16.72 | 18.27 | 18.61 |
| CCSDS | $(128, 64)$ | 4 dB | 7.68 | 6.88 | – | – | 7.88 | 7.93 | **8.28** |
| | | 5 dB | 11.88 | 10.90 | – | – | 12.62 | 12.51 | **12.87** |
| | | 6 dB | 17.50 | 15.90 | – | – | 17.08 | 17.54 | **17.75** |
| MacKay | $(96, 48)$ | 4 dB | 7.97 | 7.38 | – | – | 8.34 | 8.28 | **8.56** |
| | | 5 dB | 11.77 | 10.72 | – | – | 11.93 | 11.96 | **12.50** |
| | | 6 dB | 15.52 | 14.83 | – | – | 15.43 | **16.04** | 15.89 |

### B.2 Ablation on Hybrid method

The design of the hybrid method follows from the ablation experiment that tests different combinations of our three models, with results shown in Table 7.

Table 6: Ablation study of DEER across different codes and SNRs. Higher values are better.

| | | BCH $(63, 36)$ | | | Polar $(128, 64)$ | | | Polar $(64, 32)$ | | | CCSDS $(128, 64)$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ablation | Setting | 4 dB | 5 dB | 6 dB | 4 dB | 5 dB | 6 dB | 4 dB | 5 dB | 6 dB | 4 dB | 5 dB | 6 dB |
| **DEER** | | 5.863 | 8.123 | 10.904 | 10.443 | 13.627 | 16.150 | 7.890 | 10.451 | 13.767 | 8.386 | 13.580 | 18.137 |
| Noise scale | $C = 0$ | 5.913 | 8.154 | 10.873 | 10.451 | 13.518 | 15.889 | – | – | – | – | – | – |
| | $C = 0.1$ | 5.889 | 8.151 | 11.187 | 10.364 | 13.354 | 15.966 | – | – | – | – | – | – |
| Embedding | Additive | 5.913 | 8.108 | 11.068 | 10.189 | 13.634 | 16.261 | – | – | – | – | – | – |
| | MLP | 5.906 | 8.060 | 10.926 | 10.490 | 13.710 | 16.796 | – | – | – | – | – | – |
| | No SNR | 5.843 | 8.204 | 10.916 | 10.255 | 13.552 | 16.666 | 7.912 | 10.551 | 13.931 | 8.800 | 13.863 | 17.474 |
| | No positional | 5.782 | 7.982 | 10.800 | 2.969 | 3.485 | 4.104 | 7.864 | 10.516 | 13.670 | 5.511 | 6.665 | 7.549 |
| | DDECC + noise emb. | 4.984 | 6.850 | 9.332 | 7.001 | 9.953 | 13.519 | 6.613 | 9.020 | 11.961 | 7.707 | 12.036 | 16.191 |
| | DDECC | 5.110 | 7.090 | 9.820 | 9.110 | 12.900 | 16.600 | 6.930 | 9.510 | 12.790 | 7.810 | 12.480 | 17.660 |
| Inference steps | 1 | 5.537 | 7.641 | 10.677 | 9.510 | 13.299 | 16.103 | 7.824 | 10.251 | 13.769 | 7.881 | 12.617 | 17.078 |
| | 4 | 5.713 | 7.869 | 10.938 | 10.131 | 13.680 | 15.840 | 8.021 | 10.620 | 13.946 | 8.355 | 13.085 | 17.603 |
| | 16 | 5.839 | 8.074 | 10.982 | 10.403 | 13.614 | 15.453 | 7.945 | 10.468 | 13.997 | 8.656 | 13.331 | 17.536 |
| Inference schedule | Linear | 5.831 | 8.014 | 10.875 | 10.459 | 13.640 | 16.080 | 7.960 | 10.610 | 13.627 | 8.634 | 13.505 | 17.638 |
| Parity-check | Original form | 5.535 | 7.826 | 10.903 | 10.432 | 13.766 | 16.095 | 7.829 | 10.545 | 13.781 | – | – | – |
| Training schedule | Uniform $\eta$ | 4.621 | 6.136 | 8.447 | 5.523 | 7.823 | 10.350 | 3.343 | 3.845 | 4.336 | 3.889 | 4.721 | 5.446 |
| Batch size | 128 | 5.486 | 7.507 | 10.270 | 9.233 | 12.483 | 14.646 | 7.888 | 10.496 | 13.321 | 8.594 | 13.302 | 16.385 |
| | 1024 | 6.043 | 8.451 | 11.213 | 10.531 | 13.682 | 17.000 | 7.957 | 10.555 | 13.645 | 8.715 | 13.883 | 17.444 |
| $\eta_{\min}$ | 3 dB level | 6.044 | 8.414 | 11.281 | 9.942 | 13.702 | 16.217 | 7.908 | 10.621 | 13.584 | 8.627 | 13.641 | 17.974 |

Table 7: Hybrid-order and allocation ablation. Results are reported as $-\ln(\mathrm{BER})$, where higher values indicate better decoding performance. Forward schedules begin with DEER, whereas reverse schedules apply the same model allocation in the opposite order. The best result for each code and SNR is shown in bold.

| **Code** | **Hybrid schedule** | **4 dB** | **5 dB** | **6 dB** |
|---|---|---|---|---|
| BCH $(63, 36)$ | Forward 1:3 | 5.96 | 8.32 | 11.41 |
| | Forward 1:2:1 | **6.06** | **8.47** | 11.43 |
| | Forward 1:1:2 | 6.04 | 8.45 | **11.51** |
| | Reverse 1:3 | 5.66 | 7.77 | 10.92 |
| | Reverse 1:2:1 | 5.80 | 8.01 | 11.02 |
| | Reverse 1:1:2 | 5.76 | 7.95 | 10.87 |
| Polar $(64, 32)$ | Forward 1:3 | 8.00 | 10.43 | 13.88 |
| | Forward 1:2:1 | 7.92 | 10.41 | 13.90 |
| | Forward 1:1:2 | 7.94 | 10.48 | 13.94 |
| | Reverse 1:3 | 7.83 | 10.67 | 13.80 |
| | Reverse 1:2:1 | 7.87 | **10.70** | **14.07** |
| | Reverse 1:1:2 | **8.02** | 10.67 | 13.82 |
| Polar $(128, 64)$ | Forward 1:3 | 10.47 | 13.84 | 16.39 |
| | Forward 1:2:1 | 10.67 | 13.97 | 16.37 |
| | Forward 1:1:2 | **10.75** | **13.98** | **17.44** |
| | Reverse 1:3 | 8.90 | 13.11 | 16.61 |
| | Reverse 1:2:1 | 9.10 | 13.11 | 16.58 |
| | Reverse 1:1:2 | 9.29 | 13.08 | 16.60 |
| CCSDS $(128, 64)$ | Forward 1:3 | 8.64 | 13.62 | 17.54 |
| | Forward 1:2:1 | 8.78 | 13.72 | 17.84 |
| | Forward 1:1:2 | **8.78** | 13.80 | 18.08 |
| | Reverse 1:3 | 8.58 | 13.59 | **18.41** |
| | Reverse 1:2:1 | 8.70 | **13.85** | 17.60 |
| | Reverse 1:1:2 | 8.57 | 13.75 | 17.84 |

In this table, the forward direction involves the consecutive use of DEER, FAWN-L8, and FAWN-L6. The reverse denotes first applying FAWN-L6, then FAWN-L8, and DEER at the last stage. Steps taken at each stage are represented in $a : b : c$ form to represent $a$ steps using the first stage, $b$ for the second, and $c$ for the last. $b$ is omitted when it is equal to 0. 16 steps requires $4a, 4b, 4c$ steps.

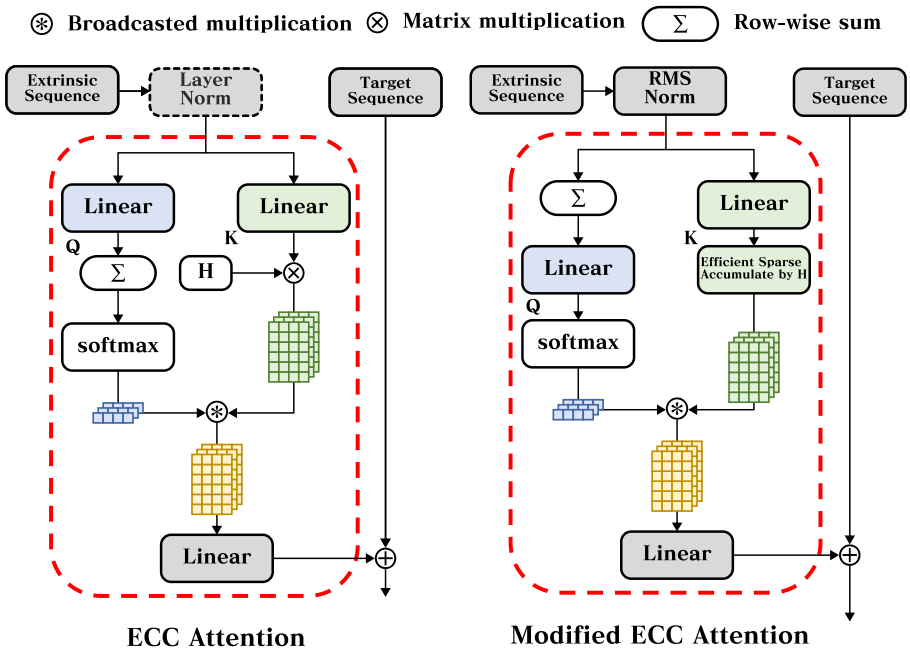


Figure 5: Architectural comparison of ECC attention by Park et al. (2026) with ours. The key differences reside in the normalization of the extrinsic sequence, the switching of computation order, and the new accumulate kernel.

From this result, we find the forward $1:1:2$ a rather competitive choice, obtaining either the best or second best performance in 10 out of 12 cases, while still being computationally cheap. We thus use this recipe for our hybrid methods.

## C Modified Attention Module

In EfficientMPT's ECC attention, a sequence was fed into the self-attention module to extract extrinsic information, then added to the target sequence, resembling the belief propagation algorithms where information from variable nodes was extracted and used to update check nodes' predictions, and the check nodes' predictions are flowed back to update variable nodes. Here, an extrinsic sequence is projected into query $Q$ and key $K$, then the query is aggregated by summation along sequence dimensions, then softmaxed to yield weights for different channels, termed global query vector $q_g$. The key is then projected using the Parity-Check matrix as a dependency mask to aggregate information from different positions of the extrinsic sequence to different entries of the target sequence. The key's different channels are then scaled by $q_g$ and then go through output projection before adding to the target sequence.

While the design is sound and reasonable, showing satisfactory decoding capacities, its implementation has followed this design faithfully, the layer normalization for the extrinsic information is not actually applied, leaving training unstable. An RMS normalization layer is therefore applied for stabilization, as it provides stabilization while being slightly faster.

We also found that the summation aggregation directly follows the linear key projections. By swapping the order of these two linear operations, input dimensions are reduced for query projection, saving large portions of projection computations.

Regarding the computation complexity, while EEC attention has eliminated the non-linear complexity of attention, its complexity was still quadratic with the matrix multiplication by $H \in \{0,1\}^{(N-K)\times N}$, thus a complexity of $O(N + N(N-K))$, with the linear part being the domi-

nant computation for short code length. Thus as shown in their paper, the actual computation grows slightly over a linear schedule.

Based on the operservation that the matrix $H$ is a sparse $0/1$ matrix fixed for the entire model during all stages, its multiplication is intrinsically a sparse gather-than-accumulate operation. A simple Triton kernel is thus compiled from the check matrix $H$. This kernel would gather and accumulate values from the nonempty positions of the check matrix, with nonzero entries being accessed sequentially, providing a complexity of $h$, where $h = \mathrm{nnz}(H)$ is the total number of nonzero elements in $H$. For LDPC codes, $h$ grows linearly with code length $N$, and other codes' systematic form PCM might also enjoy these computational savings, making a non-linear, but still non-dominant and largely reduced complexity. A kernel auto-selection policy is being introduced, which automatically executes the dense calculation kernel from EfficientMPT for shorter codes, and the new PCM-based GPU kernel when it shows more efficiency.

# D Implementation and Evaluation Details

All our models were trained using BF16 automatic mixed precision. Training employed torch.compile and the same software configuration and optimization settings across machines. Some checkpoints were trained on different compute nodes. However, the model architecture, numerical precision, batch construction, optimizer, training schedule, and compilation configuration were kept identical. Because training hardware differed between runs, training time is not used for direct model-efficiency comparisons.

All reported BER evaluations and latency measurements were conducted on the same local machine equipped with NVIDIA GeForce RTX 3090 GPUs. Evaluation used FP32 model parameters and activations with PyTorch's high float32 matrix-multiplication precision setting. All neural models were evaluated with torch. Compilation, graph tracing, projection autotuning, checkpoint loading, and warm-up executions were completed before measurements and excluded from the reported latency.

## D.1 Training and Inferencing

Our models were trained for 2,000 epochs with 1,000 minibatches per epoch using BF16 automatic mixed precision and torch.compile. DEER used a batch size of 512, while FAWN used 1,024. Both used AdamW with 20 epochs of linear warm-up followed by cosine decay to $5 \times 10^{-6}$. DEER used an initial learning rate of $1.4 \times 10^{-3}$, weight decay $0.01$, and gradient clipping at $1.0$. FAWN used $10^{-4}$, no weight decay, $(\beta_1, \beta_2) = (0.9, 0.98)$, and clipping at $0.5$.

The diffusion level was sampled uniformly in variance, $\eta_t^2 \sim \mathcal{U}(\eta_{\min}^2, 1)$, using antithetic pairs. DEER used $\eta_{\min} = 0.01$ and a reference channel SNR of 2 dB, whereas FAWN used $\eta_{\min} = 0.1$ and 3 dB. Both minimized the unweighted mean-squared error between the predicted correction and

$$\epsilon_t^\star = \mathrm{sign}(x_t) \odot (x_t - x_0)/\eta_t.$$

Parameter counts depend slightly on the code. For Polar(128,64), DEER, FAWN L6, and FAWN L8 contain approximately 2.507 M, 1.465 M, and 2.303 M parameters, respectively.

## D.2 Simulation

Transmission is simulated over a binary-input additive white Gaussian noise channel. At every tested $E_b/N_0$, random information words are encoded, modulated using BPSK, corrupted by independently generated Gaussian noise, and decoded. The principal metric is the bit-error rate over all $N$ decoded codeword positions:

$$\mathrm{BER} = \frac{\text{\# erroneous codeword bits}}{N \times \text{\# evaluated codewords}}.$$

For presentation, we often report $-\ln \mathrm{BER}$ for which larger values indicate better decoding performance. A difference of one unit corresponds to an approximately $e$-fold change in BER.

Evaluation is adaptive. Each SNR point is processed in batches until $101$ error events are collected or the maximum number of evaluated codewords ($10^7$) is reached. The minimum number of codewords is $100,352$, which is the first full batch boundary exceeding $100,000$. Low-SNR points normally terminate after collecting enough errors, whereas high-SNR points frequently reach the maximum sample count.

When at least one bit error is observed, the reported BER is the empirical ratio and remains finite. A result based on only a small number of errors is marked as having reached the sample cap because its uncertainty is correspondingly larger. If no errors are observed, the result is reported as an upper bound on BER—or equivalently a lower bound on $-\ln(\mathrm{BER})$—rather than as zero BER or infinite performance.

Confidence intervals are computed from the accumulated Bernoulli error counts. They quantify sampling uncertainty only. They do not capture variation across independently trained checkpoints, channel-model mismatch, or implementation differences. We shall provide confidence intervals along with our source code.

### D.3 Latency Measurement

Latency is measured on a single NVIDIA GeForce RTX 3090. Unless a code is too large for the standard batch, the neural benchmarks use a batch size of 2048. Tanner $(310, 128)$ uses a batch size of 256 because of its larger graph and memory footprint. Comparisons for a given code always use the same batch size.

Before timing, the model is loaded, moved to the GPU, placed in evaluation mode, and compiled. Several untimed warm-up executions are performed to trigger graph compilation, Triton code generation, cuBLAS algorithm selection, allocator initialization, and cache population. Projection autotuning is also completed before the timed region.

GPU operations are asynchronous, so wall-clock timing without synchronization would measure primarily dispatch overhead. We therefore use CUDA events, or explicit synchronization immediately around the timed region, to measure completed GPU execution. Multiple repetitions are collected after warm-up, and the arithmetic mean is reported as the batch latency.

For multistep inference, latency includes every scheduled neural forward pass and all state-update operations. Even when some codewords already satisfy their parity checks, the fixed-shape timing path executes the complete schedule so that the result reflects a stable, reproducible worst-case batch latency. Compilation, model loading, checkpoint I/O, autotuning, and the first-run warm-up are excluded.

### D.4 Classical Decoders

A series of classical decoders were added to provide better comparison. For the classical baselines, we selected a decoder appropriate to each code family: successive-cancellation list decoding with list size eight (SCL-8) for polar codes, ordered-statistics decoding of order two (OSD-2) for BCH codes, and belief propagation with at most 50 iterations (BP-50) for LDPC codes (which include the CCSDS and Mackay codes).

These decoders were implemented using NVIDIA Sionna and executed through TensorFlow with XLA compilation in FP32 precision on a single NVIDIA RTX 3090 GPU. SCL decoders were not CRC-aided; OSD's enumeration order was based on channel LLR; BP used the flooding schedule sum-product decoder without early stopping.

The parity-check and generator matrices were matched to the corresponding neural-decoder experiments to ensure that all methods operated on the same codes. The reported latency measures the complete decoding call for an already prepared batch. Data loading, model construction, compilation, and host-side initialization are excluded.

Latency was evaluated using the same batch sizes as the neural decoders: 2,048 codewords for Polar $(128, 64)$, BCH $(63, 45)$, and CCSDS $(128, 64)$, and 256 codewords for the larger Tanner (310,128) code. Compilation and warm-up executions were completed before measurement, after

Table 8: End-to-end decoding latency on an NVIDIA RTX 3090 with a batch size of 2,048. All neural decoders use compiled FP32 inference. DDECC is evaluated using its official implementation with our stated whole-batch early-stopping modification for LS rows, and the added torch.compile for fair comparison.

| **Decoder** | **Polar** $(64, 32)$ **Latency (ms)** | **BCH** $(63, 51)$ **Latency (ms)** |
|---|---|---|
| DDECC, native ($N - K$ iterations) | 2431 | 773 |
| DDECC, LS, 4 dB | 1780 | 799 |
| DDECC, LS, 5 dB | 962 | 786 |
| DDECC, LS, 6 dB | 168 | 403 |
| DEER, 1 step | 42 | 34 |
| DEER, 16 steps | 676 | 539 |
| FAWN L6, 1 step | **23** | **19** |
| FAWN L6, 16 steps | 370 | 295 |
| FAWN L8, 1 step | 31 | 25 |
| FAWN L8, 16 steps | 490 | 391 |
| Hybrid 1:1:2, 16 steps | 472 | 380 |

which repeated synchronized runs were averaged to avoid including one-time tracing and asynchronous GPU-launch overhead.

## E LATENCY COMPARISON BETWEEN DDECC AND OUR METHODS

We measured end-to-end decoding latency with a batch size of 2,048 on a single NVIDIA RTX 3090. DEER, FAWN, and the hybrid decoder used compiled FP32 inference, with ten warm-up executions followed by twenty synchronized timing repetitions. All requested diffusion steps were forcibly executed for our models as our scheme required.

The official DDECC implementation performs (N-K) decoding iterations, corresponding to 32 iterations for Polar $(64, 32)$ and 12 iterations for BCH $(63, 51)$. Its decoding loop also performs host-side bookkeeping and CPU tensor transfers between iterations, including the processing and storage of intermediate syndrome and decoding results. This implementation could likely be accelerated by keeping all intermediate operations on the GPU, eliminating transfers, and compiling or fusing the complete iterative loop.

We additionally evaluated DDECC with line search and a conservative whole-batch stopping rule. Line search evaluates multiple candidate update magnitudes and can cause the batch to reach a zero-syndrome state in fewer iterations. The modified decoder terminates only when every codeword in the batch has zero syndrome; it does not compact the active batch or terminate individual codewords independently. Consequently, the benefit depends on both SNR and batch size. At 4 dB, at least one codeword generally remains unconverged, and the decoder executes the full (N-K) iterations. At higher SNRs, particularly for Polar ((64,32)), the entire batch can converge substantially earlier. These results approximate their potential latency, and should not be attributed directly to the unmodified official DDECC implementation.

Nevertheless, under the currently available implementations, our decoders provide substantially lower latency. Even with 16 diffusion steps, FAWN L6 is approximately $4.8\times$ faster than DDECC on Polar $(64, 32)$ and $2.7\times$ faster on BCH $(63, 51)$. The one-step FAWN L6 decoder is approximately $77\times$ and $42\times$ faster, respectively. The advantage may become more pronounced for longer codes because DDECC performs $N - K$ sequential iterations, whereas the number of inference steps in DEER and FAWN remains fixed.

We note that our heaviest model, DEER, is faster than DDECC at all noise levels. At $4$ and $5$ dB, our $16$-step diffusion has lower latency than the LS version of DDECC, and at $6$ dB, as our schemes require only $4$ steps instead of $16$ steps, the overall latency shall still remains slightly lower than DDECC.