# Consistency Flow Model Achieves One-step Denoising Error Correction Codes

**Haoyu Lei**[1], **Chin Wa Lau**[2], **Kaiwen Zhou**[3], **Nian Guo**[2], **Farzan Farnia**[1]

[1] The Chinese University of Hong Kong, Computer Science & Engineering Department
[2] Huawei Technologies Co., Ltd., Theory Lab
[3] The Chinese University of Hong Kong, System Engineering & Engineering Management Department

## Abstract

Error Correction Codes (ECC) are fundamental to reliable digital communication, yet designing neural decoders that are both accurate and computationally efficient remains challenging. Recent denoising diffusion decoders with transformer backbones achieve state-of-the-art performance, but their iterative sampling limits practicality in low-latency settings. We introduce the *Error Correction Consistency Flow Model (ECCFM)*, an architecture-agnostic training framework for high-fidelity one-step decoding. By casting the reverse denoising process as a Probability Flow Ordinary Differential Equation (PF-ODE) and enforcing smoothness through a differential time regularization, ECCFM learns to map noisy signals along the decoding trajectory directly to the original codeword in a single inference step. Across multiple decoding benchmarks, ECCFM attains lower bit-error rates (BER) than autoregressive and diffusion-based baselines, with notable improvements on longer codes, while delivering inference speeds up from 30x to 100x faster than denoising diffusion decoders.

## Introduction

Error Correction Codes (ECC) play a central role in modern digital communications and have been essential in a wide range of applications, including wireless communication and data storage. The core task of an ECC decoder is to recover a message from a received signal corrupted by noise during transmission. Recently, inspired by the great success of deep neural networks in domains such as computer vision (He et al. 2016), generative modeling (Goodfellow et al. 2020; Ho, Jain, and Abbeel 2020; Song et al. 2020), and natural language processing (Vaswani et al. 2017; Brown et al. 2020), neural network-based ECC decoders were introduced, which have been shown to be capable of improving the performance scores of conventional, problem-specific algorithms such as Belief Propagation (BP)(Richardson and Urbanke 2002) and Min-Sum (MS)(Fossorier, Mihaljevic, and Imai 1999).

The existing neural decoders can be categorized into two groups. Early model-based approaches (Lugosch and Gross 2017; Nachmani and Wolf 2019; Zhu et al. 2020; Dai et al. 2021; Kwak et al. 2022) achieved successful results by integrating neural networks into conventional decoding algorithms. However, their reliance on problem-specific structures can limit their applicability as a general-purpose decoder. To address this limitation and train a general-purpose decoder, model-free decoders enable the extension of a general neural bet architecture without prior knowledge of the decoding algorithm. While the early proposals applying fully connected neural net architectures (Gruber et al. 2017; Cammerer et al. 2017; Kim et al. 2018) could lead to overfitting, the preprocessing techniques that utilize magnitude and syndrome vectors (Bennatan, Choukroun, and Kisilev 2018) have been effective in reducing the error. These improvements have led to efficient transformer-based decoders (Choukroun and Wolf 2022b), which leverage the self-attention mechanism to improve the numerical performance on short block codes. The auto-regressive method in transformer-based decoders has been improved in several recent works (Choukroun and Wolf 2024a,b). Notably, the recent work by (Park et al. 2025) integrates cross-attention between magnitude and syndrome, resulting in the state-of-the-art performance in the decoding task.

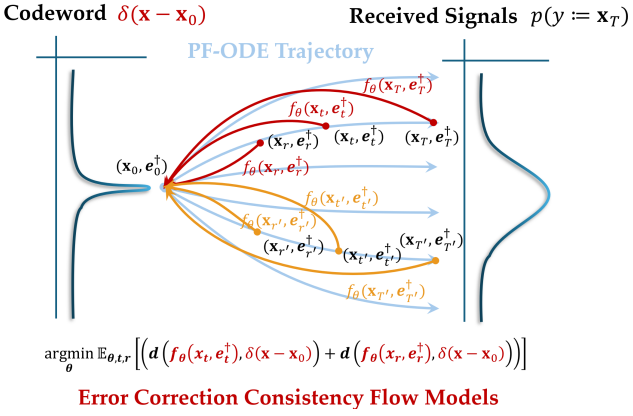



Figure 1: Illustration of our proposed Error Correction Consistency Flow Model (ECCFM). ECCFM learns to map received signals from the trajectories to a single, consistent codeword prediction $\mathbf{x}_0$, represented by a $\delta$ function.

In parallel, the remarkable success of diffusion generative models (Ho, Jain, and Abbeel 2020; Song et al. 2020) across various domains (Rombach et al. 2022; He et al. 2022; Lou, Meng, and Ermon 2023; Chen et al. 2024; Nie et al. 2025) has inspired a new direction for deep learning-based ECC. Diffusion models train a noise estimator to gradually denoise

a noisy input data and reverse a forward noise manipulation process, generating high-quality samples by an iterative application of a neural net denoiser to an input Gaussian noise. The DDECC method (Choukroun and Wolf 2022a) extends the denoising diffusion framework to ECC, naturally modeling the AWGN channel as a forward diffusion process. In this setup, a time-dependent transformer learns to denoise the received signal iteratively, recovering the original codeword. This framework provides complementary gains over auto-regressive methods and establishes a new state-of-the-art in terms of Bit Error Rates (BER), particularly for long codes and low SNR, due to its iterative refinement denoising process.

However, the iterative denoising of DDECC has introduced a new challenge: the multi-step sampling process incurs significant computational overhead and high latency, reducing the practicality of the approach in real-world applications. This motivates the following question: Can a decoder maintain the performance achieved by the denoising diffusion frameworks at a lower latency that meets the latency requirements of several communication settings?

In this work, we aim to address this question by proposing Error Correction Consistency Flow Models (ECCFM), an architecture-agnostic training framework designed to construct high-fidelity, one-step denosing decoders. Inspired by recent advances in consistency models (Song et al. 2023), we formulate the decoding process as a Probability-Flow Ordinary Differential Equation (PF-ODE), which seeks to learn an optimal trajectory from the noisy signal distribution to the clean codeword distribution as shown in Figure 1. However, a key challenge in the ECC setting is that the noise level cannot be measured by a continuous timestep as in image generation; instead, it is indicated by the sum of syndrome error in (Choukroun and Wolf 2023). A direct adaptation of consistency models struggles to learn this highly non-smooth decoding trajectory.

To overcome the mentioned challenge, we propose a differential time condition, using a soft-syndrome formulation from (Lau et al. 2025) to regularize the reverse ODE process. This ensures the decoding trajectory is smooth for single-step mapping. Building upon this theoretical foundation, the ECCFM decoder is trained directly with a consistency objective and a soft-syndrome regularization term, enabling it to map any noisy received signal to the estimated original codeword in a single, efficient inference step.

We demonstrate the effectiveness of ECCFM through several numerical experiments on a diverse set of standard codes, including BCH, Low-Density Parity-Check (LDPC), Polar, MacKay, and CCSDS. The results show that ECCFM can improve Bit-Error-Rate (BER) compared to leading auto-regressive methods, exhibiting particularly strong gains on longer codes with code lengths above 200. Notably, it achieves an inference speedup of 30x to 100x over the existing iterative denoising diffusion methods in a single step, while maintaining comparable decoding performance. In our numerical evaluation, ECCFM offers a model-free training diagram for building ECC decoders that achieves near state-of-the-art performance with low latency, and requires only a single-step inference suited for most real-world applications.

## Preliminaries

**Error Correction Codes.** In the error correction code setting, we consider a linear codebook $\mathcal{C}$, defined by a $k \times n$ generator matrix $G$ and an $(n-k) \times n$ parity-check matrix $H$. Note that these matrices satisfy $GH^\top = 0$ over the binary field $\mathbb{F}_2$. The encoder maps a message $m \in \{0,1\}^k$ to an $n$-bit codeword $x \in \mathcal{C} \subset \{0,1\}^n$ via the linear transformation $x = mG$. The codeword $x$ is then modulated using Binary Phase-Shift Keying (BPSK), where $0 \mapsto +1$ and $1 \mapsto -1$, resulting in the signal $x_s \in \{-1,+1\}^n$. We suppose this signal is transmitted over an Additive White Gaussian Noise (AWGN) channel. The received signal $y$ is given by: $y = x_s + z$, where the noise vector $z$ is sampled from an isotropic Gaussian distribution, $z \sim \mathcal{N}(0, \sigma^2 I_n)$.

The objective of a decoder is to estimate the original codeword $\hat{x}$ from the noisy signal $y$. An essential tool for error detection is the syndrome, calculated from a hard-demodulated formulation of the received signal, $y_b$, performed as $y_b = \text{bin}(\text{sign}(y))$. Here $\text{sign}(y)$ is $+1$ for $y \geq 0$ and $-1$ otherwise, and the bin function maps $\{-1,+1\}$ back to $\{1,0\}$. The syndrome is then computed as $s = Hy_b^\top$. An error is detected if the syndrome is a non-zero vector (i.e., $s(y) \neq 0$). Following the pre-processing technique proposed by (Bennatan, Choukroun, and Kisilev 2018), the input vector to the neural network is $[|y|, s(y)]$ with length $n+(n-k)$ to avoid overfitting.

**Denoising Diffusion Error Correction Codes.** Diffusion Models (DMs) (Ho, Jain, and Abbeel 2020; Song and Ermon 2019; Song et al. 2020) are generative models that generate samples from a target data distribution, $p_{\text{data}}(x_0)$ by reversing a predefined forward noising process (Sohl-Dickstein et al. 2015). In the forward diffusion process, a data sample $x_0$ is gradually perturbed with Gaussian noise over a continuous time interval $t \in [0,T]$. The application of diffusion models to error correction was pioneered by DDECC (Choukroun and Wolf 2023). Its core insight is to model the transmission of a BPSK-modulated codeword $\mathbf{x}_0 \in \{-1,1\}^n$ over an AWGN channel as the forward diffusion process. A received signal $y$ is treated as a noisy sample $\mathbf{x}_t$ at a specific timestep $t$, where the noise schedule is designed to match the channel's characteristics. This forward process is described as:

$$y := \mathbf{x}_t = \mathbf{x}_0 + \sqrt{\bar{\beta}_t}\epsilon, \tag{1}$$

where $\epsilon \sim \mathcal{N}(0, I)$, and the cumulative noise variance $\bar{\beta}_t = \sum_{i=1}^{t} \beta_i$ corresponds to the channel's noise level $\sigma^2$. Decoding is then performed via an iterative reverse denoising process, starting with the received signal $y := \mathbf{x}_t$ and applying the denoising update rule for multiple steps, with a trained denoising network $\epsilon_\theta(\cdot,\cdot)$ predicting the multiplicative noise:

$$\mathbf{x}_{t-1} = \mathbf{x}_t - \frac{\sqrt{\bar{\beta}_t}\beta_t}{\bar{\beta}_t + \beta_t}(\mathbf{x}_t - \text{sign}(\mathbf{x}_t)\epsilon_\theta(\mathbf{x}_t, t)), \tag{2}$$

A key innovation in DDECC is its conditioning mechanism. In the ECC domain, the number of parity check errors (syndrome sum), $e_t = \sum_i s(y)_i$, serves as a direct measure of

the noisy level in a received signal $y$. Therefore, DDECC conditions its denoising network $\epsilon_\theta$ on the sum of syndrome error $e_t$ instead of a timestep $t$, making the diffusion models adapted to the structure of the error correction problem. The denoising network is trained to learn the hard prediction of the multiplicative noise with a Binary Cross-Entropy (BCE) loss:

$$\mathcal{L}(\theta) = -\mathbb{E}_{e_t,\mathbf{x}_0,\epsilon} \log(\epsilon_\theta(\mathbf{x}_0 + \sqrt{\bar{\beta}_t}\epsilon, e_t), \tilde{\epsilon}_b), \quad (3)$$

where $\tilde{\epsilon}_b = \text{bin}(\mathbf{x}_0(\mathbf{x}_0 + \sqrt{\bar{\beta}_t}\epsilon))$ denotes the target binary multiplicative noise.

## Consistency Flow for One-step decoding

### Error Correction Consistency Flow Property

Despite its impressive decoding performance, a drawback of the DDECC algorithm is the computational overhead during inference due to its iterative denoising mechanism. Inspired by the recent success of Consistency Models (CMs) (Song et al. 2023; Song and Dhariwal 2023) in image generation, we propose the first approach for training consistency models for Error Correction Codes (ECC). Consistency Models (CMs) (Song et al. 2023) were introduced to overcome the inference computational costs by enabling fast, one-step generation. The core principle is that: any two points $(\mathbf{x}_t, t)$ and $(\mathbf{x}_r, r)$ on the same PF-ODE trajectory should map to the same origin point $\mathbf{x}_0$. CMs build upon Eq. 14 and learn a function $f_\theta(\mathbf{x}_t, t)$ that directly estimates the *trajectory* from noisy data to clean data with a single step:

$$f_\theta(\mathbf{x}_t, t) = \mathbf{x}_0, \quad (4)$$

The training objective of CMs is to enforce the self-consistency property across a discrete time steps. The continuous time interval $[0, T]$ is discretized into $N-1$ sub-intervals, defined by timesteps $1 = t_1 < \cdots < t_N = T$. The model is then trained to minimize the following loss, which enforces that the model's output remains consistent at adjacent points on the same PF-ODE trajectory:

$$\mathcal{L}_{\text{Standard-CM}}(\theta) := \mathbb{E}[w(t)d\left(f_\theta(\mathbf{x}_t, t), f_\theta(\mathbf{x}_r, r)\right)], \quad (5)$$

Here, $f_\theta$ is the consistency network being trained and $w(t)$ denotes the time schedule. This function is defined by two fundamental properties proposed by (Song et al. 2023; Song and Dhariwal 2023): **1) Boundary Condition**: At timestep $t = 0$, the function is the identity: $f_\theta(\mathbf{x}_0, 0) = \mathbf{x}_0$. **2) Self-Consistency**: For any two points $(\mathbf{x}_t, t)$ and $(\mathbf{x}_r, r)$ on the same trajectory, the function yields the same output: $f_\theta(\mathbf{x}_t, t) = f_\theta(\mathbf{x}_r, r)$.

First, we highlight that these properties are naturally inherent to ECC tasks. The received signal $y$ is treated as a noisy sample $\mathbf{x}_t$ from a clean codeword $\mathbf{x}_0$ during the forward diffusion process, and then the Boundary Condition trivially holds. The Self-Consistency property is also intrinsic to the decoding task. For any ground-truth codeword $\mathbf{x}_0$, all possible noisy signals $y$ with the same channel noise $z$ and different noise levels $t$ belong to trajectories that all lead back to $\mathbf{x}_0$. This highlights the motivation for designing a consistency function in decoding tasks.

Unlike generation tasks, which aim to sample from a target data distribution $p_{\text{data}}$, the goal in ECC is to decode a single, ground-truth codeword $\mathbf{x}_0$ corresponding to a given received noisy signal $y$. The ideal target distribution for the decoder is therefore a point mass at the correct codeword, which we model conceptually as a Dirac delta function, $\delta(\mathbf{x} - \mathbf{x}_0)$. Based on this, we propose that a consistency-based decoder should satisfy the following Error Correction Consistency Flow condition:

**Error Correction Self-consistency.** Given a trajectory $\{y := \mathbf{x}_t\}_{t\in[0,T]}$, we learn the consistency function as $f : (\mathbf{x}_t, t; \theta) \mapsto \delta(\mathbf{x} - \mathbf{x}_0)$, holding the Error Correction Consistency Flow property: for a given ground-truth codeword $\mathbf{x}_0$, all points $(\mathbf{x}_t, t)$ on any trajectory originating from $\mathbf{x}_0$ map directly back to it, i.e. $f_\theta(\mathbf{x}_t, t) = \delta(\mathbf{x} - \mathbf{x}_0), \quad \forall t \in [0, T]$. This implies that for any two noisy signal $\mathbf{x}_t$ and $\mathbf{x}_r$ derived from the same $\mathbf{x}_0$, their consistency function outputs must be identical and correct: $f_\theta(\mathbf{x}_t, t) = f_\theta(\mathbf{x}_r, r) = \delta(\mathbf{x} - \mathbf{x}_0), \forall t, r \in [0, T]$.

To train a consistency model $f_\theta$ that achieves this property, a naive adaptation of the vanilla consistency loss would be to minimize the distance between two noisy signal from the same trajectory similar to Eq. 5. However, this standard consistency objective is indirect; it only enforces relative consistency between outputs. In decoding tasks, the ground truth $\mathbf{x}_0$ is known during training, and consistency models are optimized to decode the clean codeword. We leverage this property by proposing the Error Correction Consistency Flow Loss (EC-CM), which directly minimizes the distance of each estimation to the ground truth $\mathbf{x}_g$ and optimizes for the upper bound of the standard consistency loss by *Triangle Inequality*:

$$\begin{aligned}\mathcal{L}_{\text{EC-CM}}(\theta) := \mathbb{E}[&d\left(f_\theta(\mathbf{x}_t, t), \delta(\mathbf{x} - \mathbf{x}_0)\right) \\ &+ d\left(f_\theta(\mathbf{x}_r, r), \delta(\mathbf{x} - \mathbf{x}_0)\right)] \\ &\geq \mathcal{L}_{\text{Standard-CM}}(\theta)\end{aligned} \quad (6)$$

It provides a tighter and direct training objective in decoding tasks. Instead of only encouraging two noisy estimates to agree with each other, it forces both to agree with the correct codeword, directly optimizing for the Error Correction Consistency property. This fundamental change requires all decoding trajectories mapping to their origin codeword as demonstrated in Figure 1, and the learned solution distribution is expected to center on $\mathbf{x}_0$.

### Differential Consistency Condition for Smooth Trajectory

The training objective of a consistency model is to learn a function $f_\theta(\mathbf{x}_t, t)$ that maps any point on a trajectory back to its origin $\mathbf{x}_0$. This is enforced by both satisfying the self-consistency property and the boundary condition, which is rooted in the differential equation $\frac{\mathrm{d}f}{\mathrm{d}t} = 0$. Following Eq. 10 and 11 in (Geng et al. 2024), the consistency function $f_\theta$ is parameterized to satisfy these conditions:

$$f_\theta(\mathbf{x}_t, t) = \mathbf{x}_0 \Leftrightarrow \frac{\mathrm{d}f}{\mathrm{d}t} = 0, f_\theta(\mathbf{x}_0, 0) = \mathbf{x}_0, \quad (7)$$

In practice, this differential form is discretized for training using a *finite-difference approximation*, by dividing the time

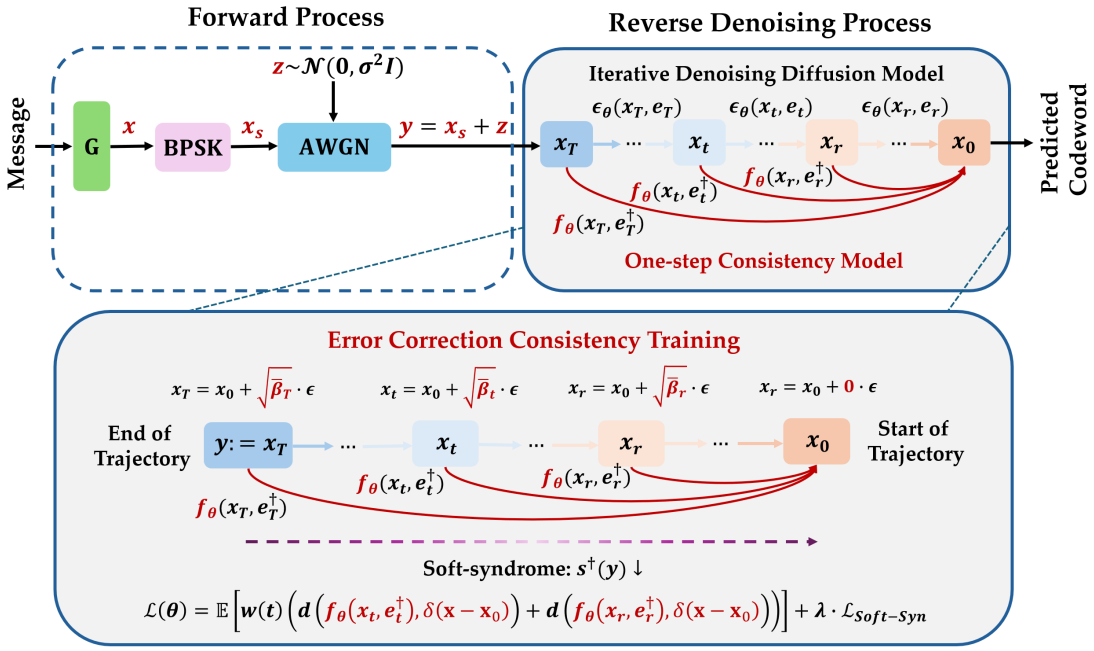



Figure 2: Training Dynamics from iterative denoising to 1-step consistency decoding. DDECC's iterative diffusion denoising learns a noise predictor, $\epsilon_\theta(\cdot, e_t)$, requiring a multi-step iterative process to reverse the noise and decode the codeword. Our ECCFM, $f_\theta(\cdot, e^\dagger)$, directly learns the mapping from any noisy signal to the original clean codeword. By using the smooth soft-syndrome condition ($e^\dagger$), it achieves successful decoding in a single step.

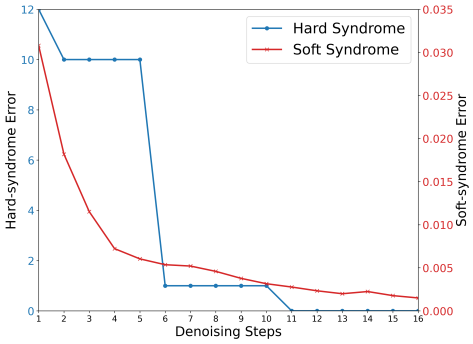



Figure 3: Decoding trajectories for models conditioned on Hard Syndrome versus Soft Syndrome on a POLAR(64,48) code. The soft-syndrome conditioning results in a smoother path to a valid codeword.

horizon into $N-1$ sub-intervals $1 = t_1 < \cdots < t_N = T$:

$$0 = \frac{\mathrm{d}f}{\mathrm{d}t} \approx \frac{f_\theta(\mathbf{x}_t, t) - f_\theta(\mathbf{x}_r, r)}{t - r}, \tag{8}$$

where $\mathrm{d}t = \Delta t = t - r$, $t > r \geq 0$.

A critical problem arises when applying the consistency framework to ECC decoding tasks: the time variables $t$ and $r$, which represent noise levels, are not directly observable from the received signals. A seemingly natural solution, proposed in DDECC (Choukroun and Wolf 2023), is to use the sum of syndrome error, $e_t$, as a measurement of the noise level, since an error count of zero ($e_t = 0$) indicates a valid codeword. The hard syndrome is computed as $s = Hy_b^\top$, where $H$ is the parity-check matrix. The sum of syndrome error, $e_t = \sum_i s(\mathbf{x}_t)_i$, is then the sum of binary syndrome bits.

However, naively replacing the continuous time variables $t$ and $r$ in Eq. 8 with their discrete, integer-valued counterparts $e_t$ and $e_r$ violates the core assumption of a smooth, differential trajectory. As we demonstrate in Figure 3, the trajectory defined by the syndrome error is highly non-smooth and clustered: A small change in the noisy signal can cause an abrupt jump in the syndrome error count, invalidating the finite-difference approximation; And it is common for two different noisy signals, $\mathbf{x}_t$ and $\mathbf{x}_r$, to have the same syndrome error count ($e_t = e_r$), leading to instability during training.

Therefore, the discrete and non-smooth nature of the syndrome error makes it an unsuitable conditioning variable for

Algorithm 1: Error Correction Consistency Training

**Require:** Model $f_\theta$, parity-check matrix $\mathbf{H}$, learning rate $\eta$, syndrome weight $\lambda$, denoising steps $N$, time scaling factor $\alpha$.

**for** training batch $\mathbf{x}_0$ **do**

$t \sim \mathcal{U}\{1, \ldots, N\}$, $r = \alpha t$

$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

$\mathbf{x}_t \leftarrow \mathbf{x}_0 + \sqrt{\bar{\beta}_t} \cdot \boldsymbol{\epsilon}, \mathbf{x}_r \leftarrow \mathbf{x}_0 + \sqrt{\bar{\beta}_r} \cdot \boldsymbol{\epsilon}$

$e_t^\dagger = \mathcal{L}_{\text{Soft-syn}}(\mathbf{x}_t, H),\ e_r^\dagger = \mathcal{L}_{\text{Soft-syn}}(\mathbf{x}_r, H)$

Compute $\mathcal{L}_{\text{EC-CM}}$ according to Eq.

Compute $\mathcal{L}_{\text{Total}}$ according to Eq. 11

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{Total}}$

**end for**

standard consistency model training in ECC, which necessitates a smooth and differential measure of noise level along the denoising trajectory. We then propose replacing this discrete error sum condition with a continuous and differentiable alternative. Inspired by (Lau et al. 2025), we introduce the *soft syndrome* as the basis for the consistency noise condition. The soft syndrome, $s^\dagger$, is a fully differentiable function that leverages the log-likelihood ratios of the received signal $y$: and offers a continuous measure of how close each parity-check equation is to being fulfilled.

**Differential Soft-syndrome Error Condition.** Similar to the conventional hard sum of syndrome error, we compute the soft-syndrome error condition $e^\dagger$ for each row $j$ of the parity-check matrix as:

$$\mathcal{L}_{\text{Soft-syn}}(y, H) = e^\dagger := \frac{1}{n-k} \sum_j \log s_j^\dagger, \tag{9}$$

This soft-syndrome error condition is computed as the binary cross-entropy between the estimated syndrome and the all-zero syndrome, which requires valid codewords to satisfy all parity-check equations. We use the mean-field approximation to provide differential conditions to estimate the probability of satisfying zero-syndrome conditions in Eq. 9:

$$s_j^\dagger = \frac{1}{2} + \frac{1}{2} \prod_{\{i:H_{j,i}=1\}} \Big(2 \cdot \text{sigmoid}\big(\frac{2y_i}{\sigma^2}\big) - 1\Big), \tag{10}$$

This metric is zero if and only if the codeword is valid, yet it varies smoothly and continuously with the received signal $\mathbf{y}$. By using $e_t^\dagger$ as the time condition, we provide the consistency model with a smooth, differentiable trajectory from a noisy signal to a valid codeword, resolving the instability and degeneracy issues of the hard error sum and enabling stable training.

## Error Correction Consistency Training Dynamics

Building upon the differential time conditions via soft-syndrome in Eq. 9, we make it able to learn a smooth trajectory satisfying consistency conditions as shown in Figure 2. Given a codeword $x_s \in \{-1, +1\}^n$ modulated using BPSK, and the signal received is then perturbed with an AWGN channel $y = x_s + z = x_s \cdot \tilde{z}_s$, where $\tilde{z}_s$ denotes the multiplicative noise. We follow the standard preprocessing techniques proposed by (Bennatan, Choukroun, and Kisilev 2018), the input of the neural network is a concatenated vector representing magnitude and hard syndrome $[|y|, s(y)]$ with length $2n - k$. For the forward process in AWGN channel, we build the trajectory by adding the same Gaussian noise $\epsilon \sim \mathcal{N}(0, I)$ with a different time schedule $\sqrt{\bar{\beta}_t}$, where $t \in [0, ..., N]$ and $N$ denotes the pre-defined forward noising steps. Thus, during training, we sample different noisy signals $y_1 := \mathbf{x}_t, y_2 := \mathbf{x}_r$ with different noise levels $t \sim \mathcal{U}\{0, ..., N\}$ and $r = \alpha t$, i.e.$\mathbf{x}_t := \mathbf{x}_0 + \sqrt{\bar{\beta}_t} \cdot \epsilon$, $\mathbf{x}_r := \mathbf{x}_0 + \sqrt{\bar{\beta}_r} \cdot \epsilon$, where $\alpha \in [0, 1]$ denotes the time scaling factor. Then a consistency model $f\theta$ predicts the clean codeword. Following Eq. , we get the consistency loss for two different noisy signals $y_t, y_r$, which learns the mapping to their original clean codeword $\mathbf{x}_0$. We further add the soft-syndrome loss in Eq. 9 as the regularization term to stabilize training and propose the total loss for ECCFM:

$$\begin{aligned} \mathcal{L}_{\text{Total}}(\theta) = \mathbb{E}\Big[ w(t) \Big( & d(f_\theta(\mathbf{x}_t, e_t^\dagger), \delta(\mathbf{x} - \mathbf{x}_0)) \\ & + d(f_\theta(\mathbf{x}_r, e_r^\dagger), \delta(\mathbf{x} - \mathbf{x}_0)) \Big) \\ & + \lambda \cdot \Big( \mathcal{L}_{\text{Soft-syn}}(f_\theta(\mathbf{x}_t, e_t^\dagger), H) \\ & + \mathcal{L}_{\text{Soft-syn}}(f_\theta(\mathbf{x}_r, e_r^\dagger), H) \Big) \Big] \end{aligned} \tag{11}$$

where $d(\cdot, \cdot)$ is a distance metric, such as Binary Cross-Entropy (BCE) in this work, and $\lambda$ is a hyperparameter that weights the syndrome regularization term. Once trained according to Algorithm 1, the learned consistency function $f_\theta$ can decode the noisy received signal $y$ in a one step: $\hat{\mathbf{x}_0} = f_\theta(y, e_t^\dagger)$, as shown in Appendix 2, Algorithm 2.

## Numerical Results

**Datasets.** We evaluate our proposed ECCFM framework on the following set of standard error correction codes, including BCH, Polar, and LDPC codes (MacKay, CCSDS, and WRAN variants). Our evaluation considers multiple code lengths ($n$), rates ($k/n$), and Signal-to-Noise Ratios (SNRs), specifically $E_b/N_0$ values from 4 to 6 dB, to ensure a robust assessment of performance.

**Evaluation Metrics.** We evaluate decoding performance using two standard metrics following established benchmarks (Choukroun and Wolf 2022b, 2023; Park et al. 2025): Bit Error Rate (BER) and Frame Error Rate (FER). BER measures the fraction of individual bits that are incorrectly decoded. FER (also known as Block Error Rate, BLER) measures the fraction of entire codewords that contain one or more bit errors. Concerning the latency factor, we evaluate computational efficiency by reporting inference time and throughput (decoded samples per second).

**Baselines.** We numerically compared the results with multiple baselines for the decoding task, including: 1) Conventional BP-based decoders: BP (Bennatan, Choukroun, and Kisilev 2018) and ARBP (Nachmani and Wolf 2021). 2) Auto-regressive model-free decoders: ECCT (Choukroun and Wolf 2022b), and CrossMPT (Park et al. 2025). 3) Denoising diffusion model-free decoders: DDECC (Choukroun and Wolf 2023).

**Experimental Setup.** We reproduced the results for all model-free baselines (ECCT, CrossMPT, DDECC) by implementing them with their publication-stated hyperparameters. Our primary ECCFM model utilizes a Transformer architecture with cross-attention, using $N = 6$ layers and a hidden dimension of $d = 128$, and all the other three baselines (ECCT, CrossMPT, DDECC) apply the same model architecture to ensure fair comparison. ECCFM was trained for 1500 epochs using the Adam optimizer on a single GPU. The learning rate was managed by a cosine decay scheduler, starting at $10^{-4}$ and decreasing to $5 \times 10^{-7}$. Detailed training configurations and hyperparameter selections are provided in Appendix 4.

**Overall Performance**. Following established benchmarks (Choukroun and Wolf 2022b, 2023; Park et al. 2025),

Table 1: Performance comparison of various decoders across different codes and Signal-to-Noise Ratios ($E_b/N_0$). The results are reported in terms of $-\ln(\text{BER})$ (the higher, the better). All model-free methods use a fixed model architecture ($N = 6$, $d = 128$). Best results are shown in **bold** and the second-best results are shown in **underline**, respectively.

| Architecture | | BP-based decoders | | | | | | Model-free decoders | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Code Type | Parameters | BP | | | ARBP | | | ECCT | | | CrossMPT | | | DDECC | | | **ECCFM(Ours)** | | |
| | | **4** | **5** | **6** | **4** | **5** | **6** | **4** | **5** | **6** | **4** | **5** | **6** | **4** | **5** | **6** | **4** | **5** | **6** |
| **BCH** | (63,36) | 4.03 | 5.42 | 7.26 | 4.57 | 6.39 | 8.92 | 4.69 | 6.48 | 9.06 | 4.94 | 6.74 | 9.28 | **5.02** | 6.82 | **9.88** | 5.00 | **6.89** | 9.76 |
| | (63,45) | 4.36 | 5.55 | 7.26 | 4.97 | 6.90 | 9.41 | 5.47 | 7.56 | 10.51 | **5.73** | 7.98 | 10.80 | 5.68 | **8.08** | **11.22** | 5.70 | 8.03 | 11.04 |
| **POLAR** | (64,32) | 4.26 | 5.38 | 6.50 | 5.57 | 7.43 | 9.82 | 6.87 | 9.21 | 12.15 | 7.42 | 9.94 | 13.28 | 7.04 | 9.44 | 12.70 | **7.55** | **10.31** | **13.80** |
| | (64,48) | 4.74 | 5.94 | 7.42 | 5.41 | 7.19 | 9.30 | 6.21 | 8.31 | 10.85 | 6.36 | 8.53 | 11.09 | 5.93 | 8.00 | 10.44 | **6.56** | **8.78** | **11.52** |
| | (128,64) | 4.10 | 5.11 | 6.15 | 4.84 | 6.78 | 9.30 | 5.79 | 8.45 | 11.10 | 7.45 | 9.71 | 14.31 | 7.71 | 11.40 | 13.85 | **8.01** | **12.22** | **16.71** |
| | (128,86) | 4.49 | 5.65 | 6.97 | 5.39 | 7.37 | 10.13 | 6.29 | 8.98 | 12.82 | 7.43 | 10.80 | 15.13 | 7.61 | 10.50 | 13.88 | **7.78** | **11.21** | **16.05** |
| | (128,96) | 4.61 | 5.79 | 7.08 | 5.27 | 7.44 | 10.20 | 6.30 | 9.04 | 12.40 | 7.06 | 10.25 | 13.23 | 7.14 | 10.31 | 13.66 | **7.21** | **10.52** | **14.32** |
| **LDPC** | (121,60) | 4.82 | 7.21 | 10.87 | 5.22 | 8.31 | 13.07 | 5.12 | 8.21 | 12.80 | 5.75 | 9.42 | 15.21 | 5.42 | 9.11 | 13.82 | **6.02** | **9.94** | **15.55** |
| | (121,70) | 5.88 | 8.76 | 13.04 | 6.45 | 10.01 | 14.77 | 6.30 | 10.11 | 15.50 | 7.06 | 11.29 | 17.10 | 6.91 | 11.02 | 17.15 | **7.35** | **12.23** | **17.60** |
| | (121,80) | 6.66 | 9.82 | 13.98 | 7.22 | 11.03 | 15.90 | 7.27 | 11.21 | 17.02 | 7.87 | 12.65 | 17.72 | 7.61 | 11.89 | 16.18 | **8.25** | **13.33** | **18.69** |
| **MacKay** | (96,48) | 6.84 | 9.40 | 12.57 | 7.43 | 10.65 | 14.65 | 7.37 | 10.55 | 14.72 | 7.85 | 11.72 | 15.49 | **8.03** | **12.44** | 15.79 | 7.92 | 12.25 | **16.08** |
| **CCSDS** | (128,64) | 6.55 | 9.65 | 13.78 | 7.25 | 10.99 | 16.36 | 6.82 | 10.60 | 15.87 | 7.56 | 11.87 | 16.80 | 7.77 | 12.35 | **17.22** | **7.95** | **12.68** | 17.01 |

we conducted a decoding performance comparison measured in $-\ln(\text{BER})$. Our method was evaluated versus two classes of decoders: conventional BP-based algorithms (BP and ARBP) and model-free neural decoders (ECCT, CrossMPT, and DDECC). To ensure a fair comparison, all neural models were implemented with a fixed architecture ($N = 6$ layers, $d = 128$ hidden dimensions). Furthermore, to ensure statistical significance, each simulation was run until at least 500 error codes were observed, under a maximum of $10^7$ test instances.

As shown in Table 1, our proposed ECCFM framework consistently achieves better performance across the tested code families, including BCH, Polar, LDPC, CCSDS and MacKay with different code rates $(n, k)$. In the test cases, ECCFM achieves the best or second-best BER, performing better than autoregressive and diffusion-based neural decoders and showing considerable gain over POLAR codes. This result indicates the applicability and effectiveness of the proposed consistency-based training approach.

**Performance on Longer Codes.** To further evaluate scalability, we conducted a focused evaluation on longer codes commonly used in practical communication systems: LDPC$(n = 204, k = 102)$, LDPC$(n = 408, k = 204)$, WRAN$(n = 384, k = 320)$, and Polar$(n = 512, k = 384)$. All methods were implemented with the same model architecture ($N = 6, d = 128$). As illustrated in Figure 4, ECCFM improves upon the scores of the neural net baselines across a range of SNRs (2 dB to 6 dB), highlighting the scalability and robustness of ECCFM for those challenging decoding tasks. Additional results on other high-length codes are presented in Appendix 4, which also suggest performance gain compared to the baselines.

**Inference Time and Throughput Comparison.** A key benefit of ECCFM is its ability to perform high-fidelity decoding in only one step. To quantify this efficiency gain, we measured inference time (total seconds to decode $10^5$ samples) and throughput (samples decoded per second). As shown in Figure 5, ECCFM demonstrates a speed advantage over diffusion-based methods such as DDECC, achieving speedups of over **30x** for short codes and **100x** for medium-to-long codes. This disparity arises because diffusion models require several iterative denoising steps for inference, a computational cost that scales with code complexity as detailed in Appendix 4. Notably, ECCFM achieves decoding speeds comparable to the fastest auto-regressive baseline (CrossMPT) due to its one-step nature. Therefore, ECCFM matches the competitive performance of denoising diffusion decoders while operating at the high throughput of single-step auto-regressive decoders.

**Ablation Study: Model-Agnostic Property of ECCFM.** We discussed that ECCFM is a model-agnostic training framework, i.e., its performance could be preserved over different neural network architectures. To test this, we conducted an ablation study where we decoupled our framework from cross-attention transformer backbone conducted before. Specifically, we took the underlying architecture of the ECCT baseline and trained it using our proposed ECCFM training objective. We then compared this model directly against the original ECCT, which uses the same architecture. The results presented in Appendix 4, Table 3 show that applying the ECCFM training objective yields improvement in $-\ln(\text{BER})$ over the standard ECCT.

## Conclusions and Limitations

In this work, we introduced the Error Correction Consistency Flow Model (ECCFM), a novel training framework for obtaining the successful results of diffusion-based decoders with the low latency required for several practical applications. By reformulating the decoding task as a one-step consistency mapping and introducing a differential soft-syndrome error condition, we managed to handle non-smooth trajectories that previously hindered the application of consistency models to ECC. Our experiments demonstrate that ECCFM achieves comparable Bit-Error-

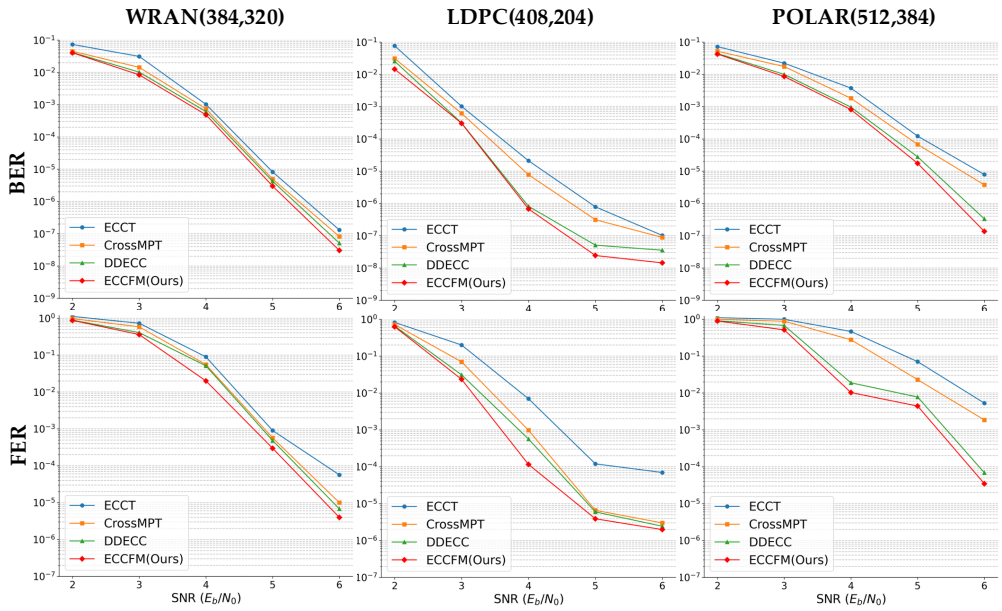



Figure 4: Performance comparison of various decoding baselines on medium-to-long block codes. The plot shows the Bit Error Rate (BER) at different Signal-to-Noise Ratios (SNRs).

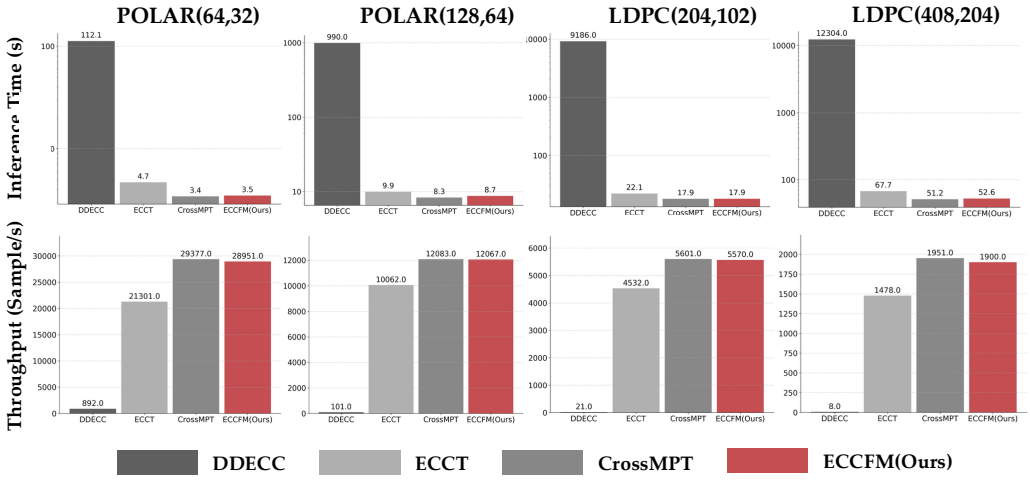



Figure 5: Comparison of Inference Time (top) and Throughput (bottom) across various decoding baselines and code types.

Rates across various standard codes, while offering a consistent inference speedup over denoising diffusion methods. Despite these successful results, our work has limitations to be addressed in future research. First, our evaluation is focused on the AWGN channel; the framework's performance and the suitability of the soft-syndrome condition on other channel models, such as fading channels, remain to be investigated. Second, the convergence rate and training efficiency of consistency models highly rely on building a smooth trajectory, which varies in different code types. Future work can explore adaptive methods for more general decoding tasks.

## References


Bennatan, A.; Choukroun, Y.; and Kisilev, P. 2018. Deep learning for decoding of linear codes-a syndrome-based approach. In *2018 IEEE International Symposium on Information Theory (ISIT)*, 1595–1599. IEEE.

Blattmann, A.; Rombach, R.; Ling, H.; Dockhorn, T.; Kim, S. W.; Fidler, S.; and Kreis, K. 2023. Align your latents: High-resolution video synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 22563–22575.

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901.

Cammerer, S.; Gruber, T.; Hoydis, J.; and Ten Brink, S. 2017. Scaling deep learning-based decoding of polar codes via partitioning. In *GLOBECOM 2017-2017 IEEE global communications conference*, 1–6. IEEE.

Chen, H.; Zhang, Y.; Cun, X.; Xia, M.; Wang, X.; Weng, C.; and Shan, Y. 2024. Videocrafter2: Overcoming data limitations for high-quality video diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 7310–7320.

Choukroun, Y.; and Wolf, L. 2022a. Denoising diffusion error correction codes. *arXiv preprint arXiv:2209.13533*.

Choukroun, Y.; and Wolf, L. 2022b. Error correction code transformer. *Advances in Neural Information Processing Systems*, 35: 38695–38705.

Choukroun, Y.; and Wolf, L. 2023. Denoising Diffusion Error Correction Codes. In *The Eleventh International Conference on Learning Representations*.

Choukroun, Y.; and Wolf, L. 2024a. A foundation model for error correction codes. In *The Twelfth International Conference on Learning Representations*.

Choukroun, Y.; and Wolf, L. 2024b. Learning linear block error correction codes. In *Proceedings of the 41st International Conference on Machine Learning*, 8801–8814.

Dai, J.; Tan, K.; Si, Z.; Niu, K.; Chen, M.; Poor, H. V.; and Cui, S. 2021. Learning to decode protograph LDPC codes. *IEEE Journal on Selected Areas in Communications*, 39(7): 1983–1999.

Dhariwal, P.; and Nichol, A. 2021. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34: 8780–8794.

Fossorier, M. P.; Mihaljevic, M.; and Imai, H. 1999. Reduced complexity iterative decoding of low-density parity check codes based on belief propagation. *IEEE Transactions on communications*, 47(5): 673–680.

Geng, Z.; Pokle, A.; Luo, W.; Lin, J.; and Kolter, J. Z. 2024. Consistency models made easy. *arXiv preprint arXiv:2406.14548*.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2020. Generative adversarial networks. *Communications of the ACM*, 63(11): 139–144.

Gruber, T.; Cammerer, S.; Hoydis, J.; and Ten Brink, S. 2017. On deep learning-based channel decoding. In *2017 51st annual conference on information sciences and systems (CISS)*, 1–6. IEEE.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

He, Y.; Yang, T.; Zhang, Y.; Shan, Y.; and Chen, Q. 2022. Latent video diffusion models for high-fidelity long video generation. *arXiv preprint arXiv:2211.13221*.

Ho, J.; Jain, A.; and Abbeel, P. 2020. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, volume 33, 6840–6851.

Karras, T.; Aittala, M.; Aila, T.; and Laine, S. 2022. Elucidating the design space of diffusion-based generative models. *Advances in neural information processing systems*, 35: 26565–26577.

Kim, H.; Jiang, Y.; Rana, R. B.; Kannan, S.; Oh, S.; and Viswanath, P. 2018. Communication Algorithms via Deep Learning. In *International Conference on Learning Representations*.

Kwak, H.-Y.; Kim, J.-W.; Kim, Y.; Kim, S.-H.; and No, J.-S. 2022. Neural min-sum decoding for generalized LDPC codes. *IEEE Communications Letters*, 26(12): 2841–2845.

Kwak, H.-Y.; Yun, D.-Y.; Kim, Y.; Kim, S.-H.; and No, J.-S. 2023. Boosting learning for LDPC codes to improve the error-floor performance. *Advances in Neural Information Processing Systems*, 36: 22115–22131.

Lau, C. W.; Shi, X.; Zheng, Z.; Cao, H.; and Guo, N. 2025. Interplay Between Belief Propagation and Transformer: Differential-Attention Message Passing Transformer. arXiv:2509.15637.

Lei, H.; Zhou, K.; Li, Y.; Chen, Z.; and Farnia, F. 2025. Boosting Generalization in Diffusion-Based Neural Combinatorial Solver via Inference Time Adaptation. *arXiv preprint arXiv:2502.12188*.

Li, Y.; Guo, J.; Wang, R.; and Yan, J. 2023. T2t: From distribution learning in training to gradient search in testing for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36: 50020–50040.

Lou, A.; Meng, C.; and Ermon, S. 2023. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*.

Lu, C.; Zhou, Y.; Bao, F.; Chen, J.; Li, C.; and Zhu, J. 2022. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *Advances in neural information processing systems*, 35: 5775–5787.

Lugosch, L.; and Gross, W. J. 2017. Neural offset min-sum decoding. In *2017 IEEE International Symposium on Information Theory (ISIT)*, 1361–1365. IEEE.

Marinkovic, M.; Piz, M.; Choi, C.-S.; Panic, G.; Ehrig, M.; and Grass, E. 2010. Performance evaluation of channel coding for Gbps 60-GHz OFDM-based wireless communications. In *21st Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, 994–998. IEEE.

Matsumine, T.; and Ochiai, H. 2024. Recent advances in deep learning for channel coding: A survey. *IEEE Open Journal of the Communications Society*.

Nachmani, E.; and Wolf, L. 2019. Hyper-graph-network decoders for block codes. *Advances in Neural Information Processing Systems*, 32.

Nachmani, E.; and Wolf, L. 2021. Autoregressive belief propagation for decoding block codes. *arXiv preprint arXiv:2103.11780*.

Nie, S.; Zhu, F.; You, Z.; Zhang, X.; Ou, J.; Hu, J.; Zhou, J.; Lin, Y.; Wen, J.-R.; and Li, C. 2025. Large language diffusion models. *arXiv preprint arXiv:2502.09992*.

Park, S.-J.; Kwak, H.-Y.; Kim, S.-H.; Kim, Y.; and No, J.-S. 2024. CrossMPT: Cross-attention message-passing transformer for error correcting codes. *arXiv preprint arXiv:2405.01033*.

Park, S.-J.; Kwak, H.-Y.; Kim, S.-H.; Kim, Y.; and No, J.-S. 2025. CrossMPT: Cross-attention Message-passing Transformer for Error Correcting Codes. In *The Thirteenth International Conference on Learning Representations*.

Podell, D.; English, Z.; Lacey, K.; Blattmann, A.; Dockhorn, T.; Müller, J.; Penna, J.; and Rombach, R. 2024. SDXL: Improving Latent Diffusion Models for High-Resolution Image Synthesis. In *The Twelfth International Conference on Learning Representations*.

Richardson, T. J.; and Urbanke, R. L. 2002. The capacity of low-density parity-check codes under message-passing decoding. *IEEE Transactions on information theory*, 47(2): 599–618.

Rombach, R.; Blattmann, A.; Lorenz, D.; Esser, P.; and Ommer, B. 2022. High-Resolution Image Synthesis with Latent Diffusion Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 10684–10695.

Sohl-Dickstein, J.; Weiss, E.; Maheswaranathan, N.; and Ganguli, S. 2015. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, 2256–2265. pmlr.

Song, J.; Meng, C.; and Ermon, S. 2020. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*.

Song, Y.; and Dhariwal, P. 2023. Improved techniques for training consistency models. *arXiv preprint arXiv:2310.14189*.

Song, Y.; Dhariwal, P.; Chen, M.; and Sutskever, I. 2023. Consistency models. In *Proceedings of the 40th International Conference on Machine Learning*, 32211–32252.

Song, Y.; and Ermon, S. 2019. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32.

Song, Y.; Sohl-Dickstein, J.; Kingma, D. P.; Kumar, A.; Ermon, S.; and Poole, B. 2020. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*.

Stein, C. 1972. A bound for the error in the normal approximation to the distribution of a sum of dependent random variables. In *Proceedings of the sixth Berkeley symposium on mathematical statistics and probability, volume 2: Probability theory*, volume 6, 583–603. University of California Press.

Sun, Z.; and Yang, Y. 2023. Difusco: Graph-based diffusion solvers for combinatorial optimization. *Advances in neural information processing systems*, 36: 3706–3731.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Zhu, H.; Cao, Z.; Zhao, Y.; and Li, D. 2020. Learning to denoise and decode: A novel residual neural network decoder for polar codes. *IEEE Transactions on Vehicular Technology*, 69(8): 8725–8738.

# Reproducibility Checklist

---

**Instructions for Authors:**
This document outlines key aspects for assessing reproducibility. Please provide your input by editing this `.tex` file directly.

For each question (that applies), replace the "Type your response here" text with your answer.

**Example:** If a question appears as

```
\question{Proofs of all novel claims
are included} {(yes/partial/no)}
Type your response here
```

you would change it to:

```
\question{Proofs of all novel claims
are included} {(yes/partial/no)}
yes
```

Please make sure to:

- Replace ONLY the "Type your response here" text and nothing else.
- Use one of the options listed for that question (e.g., **yes**, **no**, **partial**, or **NA**).
- **Not** modify any other part of the `\question` command or any other lines in this document.

You can `\input` this .tex file right before `\end{document}` of your main file or compile it as a stand-alone document. Check the instructions on your conference's website to see if you will be asked to provide this checklist with your paper or separately.

---

### 1. General Paper Structure

1.1. Includes a conceptual outline and/or pseudocode description of AI methods introduced (yes/partial/no/NA) yes

1.2. Clearly delineates statements that are opinions, hypothesis, and speculation from objective facts and results (yes/no) yes

1.3. Provides well-marked pedagogical references for less-familiar readers to gain background necessary to replicate the paper (yes/no) yes

### 2. Theoretical Contributions

2.1. Does this paper make theoretical contributions? (yes/no) yes

If yes, please address the following points:

2.2. All assumptions and restrictions are stated clearly and formally (yes/partial/no) yes

2.3. All novel claims are stated formally (e.g., in theorem statements) (yes/partial/no) yes

2.4. Proofs of all novel claims are included (yes/partial/no) yes

2.5. Proof sketches or intuitions are given for complex and/or novel results (yes/partial/no) yes

2.6. Appropriate citations to theoretical tools used are given (yes/partial/no) yes

2.7. All theoretical claims are demonstrated empirically to hold (yes/partial/no/NA) yes

2.8. All experimental code used to eliminate or disprove claims is included (yes/no/NA) yes

**3. Dataset Usage**

3.1. Does this paper rely on one or more datasets? (yes/no) yes

If yes, please address the following points:

3.2. A motivation is given for why the experiments are conducted on the selected datasets (yes/partial/no/NA) yes

3.3. All novel datasets introduced in this paper are included in a data appendix (yes/partial/no/NA) yes

3.4. All novel datasets introduced in this paper will be made publicly available upon publication of the paper with a license that allows free usage for research purposes (yes/partial/no/NA) NA

3.5. All datasets drawn from the existing literature (potentially including authors' own previously published work) are accompanied by appropriate citations (yes/no/NA) yes

3.6. All datasets drawn from the existing literature (potentially including authors' own previously published work) are publicly available (yes/partial/no/NA) yes

3.7. All datasets that are not publicly available are described in detail, with explanation why publicly available alternatives are not scientifically satisficing (yes/partial/no/NA) NA

**4. Computational Experiments**

4.1. Does this paper include computational experiments? (yes/no) yes

If yes, please address the following points:

4.2. This paper states the number and range of values tried per (hyper-) parameter during development of the paper, along with the criterion used for selecting the final parameter setting (yes/partial/no/NA) yes

4.3. Any code required for pre-processing data is included in the appendix (yes/partial/no) partial

4.4. All source code required for conducting and analyzing the experiments is included in a code appendix (yes/partial/no) partial

4.5. All source code required for conducting and analyzing the experiments will be made publicly available upon publication of the paper with a license that allows free usage for research purposes (yes/partial/no) yes

4.6. All source code implementing new methods have comments detailing the implementation, with references to the paper where each step comes from (yes/partial/no) yes

4.7. If an algorithm depends on randomness, then the method used for setting seeds is described in a way sufficient to allow replication of results (yes/partial/no/NA) yes

4.8. This paper specifies the computing infrastructure used for running experiments (hardware and software), including GPU/CPU models; amount of memory; operating system; names and versions of relevant software libraries and frameworks (yes/partial/no) yes

4.9. This paper formally describes evaluation metrics used and explains the motivation for choosing these metrics (yes/partial/no) yes

4.10. This paper states the number of algorithm runs used to compute each reported result (yes/no) yes

4.11. Analysis of experiments goes beyond single-dimensional summaries of performance (e.g., average; median) to include measures of variation, confidence, or other distributional information (yes/no) yes

4.12. The significance of any improvement or decrease in performance is judged using appropriate statistical tests (e.g., Wilcoxon signed-rank) (yes/partial/no) yes

4.13. This paper lists all final (hyper-)parameters used for each model/algorithm in the paper's experiments (yes/partial/no/NA) yes

## Preliminary on Diffusion Generative Models

**Diffusion Models.** Diffusion Models (DMs) (Ho, Jain, and Abbeel 2020; Song and Ermon 2019; Song et al. 2020) are generative models that generate samples from a target data distribution, $p_{\text{data}}(x_0)$ by reversing a predefined forward noising process (Sohl-Dickstein et al. 2015). In the forward diffusion process, a data sample $x_0$ is gradually perturbed with Gaussian noise over a continuous time interval $t \in [0, T]$. This forward process can be mathematically described as adding noise to obtain a noisy data point $\mathbf{x}_t = \sqrt{\alpha_t}\mathbf{x}_0 + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_t$, where $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ is standard Gaussian noise and $\alpha_t \in [0, 1]$ monotonically decreases with time step $t$ to control the noise level. Denoising Diffusion Probabilistic Models (DDPMs) (Ho, Jain, and Abbeel 2020) $\boldsymbol{\epsilon}_\theta : \mathcal{X} \times [T] \mapsto \mathcal{X}$ is trained to predict the noise $\boldsymbol{\epsilon}_t$ at each time step $t$, also learn the *score function* of $p_t(\mathbf{x_t})$ (Song and Ermon 2019; Song et al. 2020):

$$\min_\theta \mathbb{E}_{\mathbf{x}_t, \boldsymbol{\epsilon}_t, t}\left[\|\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) - \boldsymbol{\epsilon}_t\|_2^2\right] = \min_\theta \mathbb{E}_{\mathbf{x}_t, \boldsymbol{\epsilon}_t, t}\left[\left\|\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) + \sqrt{1-\alpha_t}\underbrace{\nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t)}_{\text{Score Function}}\right\|_2^2\right], \tag{12}$$

During inference, samples can be generated by solving the reverse-time SDE starting from $t = T$ to $t = 0$. Crucially, there exists a corresponding deterministic process, the *probability flow ODE* (PF-ODE), whose trajectories share the same marginal distributions $p_t(x_t)_{t\in[0,T]}$ as the SDE (Song et al. 2020). The formulation of PF-ODE can be described and simplified as following (Karras et al. 2022; Song et al. 2023):

$$\mathrm{d}\mathbf{x}_t = -\dot{\sigma}(t)\sigma(t)\nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t)\mathrm{d}t, \tag{13}$$

where $\epsilon_\theta(\mathbf{x}_t, t)$ is the learned time-dependent neural network $\epsilon_\theta(\mathbf{x}_t, t)$, known as the denoiser. , is trained to approximate this expectation: $\epsilon_\theta(x_t, t) \approx \mathbb{E}[x_0|x_t]$. By substituting this approximation and adopting the common noise schedule $\sigma(t) = t$ following (Karras et al. 2022; Song et al. 2023), the PF-ODE simplifies to:

$$\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t} = -t\nabla\mathbf{x}_t\log p_t(\mathbf{x}_t) = \frac{\mathbf{x}_t - \epsilon_\theta(\mathbf{x}_t, t)}{t}, \tag{14}$$

Sample generation of diffusion models is performed by solving this PF-ODE backwards in time from $t = T$ to $t = 0$, starting from a sample drawn from the prior Gaussian distribution, $\mathbf{x}_T \sim \mathcal{N}(0, \sigma(T)^2 I)$. This requires a numerical ODE solver (e.g., Euler (Song and Ermon 2019; Song et al. 2020) or Heun (Karras et al. 2022)) to obtain a *solution trajectory* $\{\hat{\mathbf{x}}_t\}_{t\in[0,T]}$ that transforms noise into a data sample.

**Consistency Models (CMs).** A major concern of DMs is the slow sampling process, which requires sequential calculation of the denoiser $\epsilon_\theta$. Consistency Models (CMs) (Song et al. 2023) were introduced to overcome this by enabling fast, 1-step generation. The core principle is the self-consistency property: any two points $(\mathbf{x}_t, t)$ and $(\mathbf{x}_r, r)$ on the same PF-ODE trajectory should map to the same origin point, $\mathbf{x}_0$. CMs build upon Eq. 14 and learn a function $f_\theta(\mathbf{x}_t, t)$ that directly estimates the *trajectory* from noisy data to clean data with a single step:

$$f_\theta(\mathbf{x}_t, t) = \mathbf{x}_0, \tag{15}$$

The training objective of CMs is to enforce the consistency property across a discrete set of time steps. The continuous time interval $[0, T]$ is discretized into $N-1$ sub-intervals, defined by timesteps $1 = t_1 < \cdots < t_N = T$. The model is then trained to minimize the following loss, which enforces that the model's output is consistent for adjacent points on the same ODE trajectory:

$$\arg\min_\theta \mathbb{E}[w(t_i)d(f_\theta(\mathbf{x}_{t_{i+1}}, t_{i+1}), f_{\theta-}(\tilde{\mathbf{x}}_{t_i}, t_i))], \tag{16}$$

Here, $f_\theta$ is the network being trained, while $f_{\theta-}$ is an exponential moving average (EMA) of $f_\theta$'s past samples. The term $\tilde{\mathbf{x}}_{t_i} = \mathbf{x}_{t_{i+1}} - (t_i - t_{i+1})t_{i+1}\nabla_{\mathbf{x}_{t_{i+1}}}\log p_{t_{i+1}}(\mathbf{x}_{t_{i+1}})$ is obtained by taking a single ODE solver step backwards from $\mathbf{x}_{t_{i+1}}$ using the score function. This training process can be performed in two ways: by distilling knowledge from a pre-trained diffusion model, known as Consistency Distillation (CD), or by training from scratch, known as Consistency Training (CT). However, training CMs is difficult and resource-intensive. It requires a carefully designed curriculum for the number of discretization steps $N$ to ensure stabilized training. The follow-up works improved the vanilla CMs, such as iCT (Song and Dhariwal 2023), which proposed enhanced metrics and schedulers, and ECT (Geng et al. 2024), which uses "pre-training diffusion + consistency tuning" to stabilize learning.

## Related Works

**Neural Network-based ECC Decoders.** Neural network-based decoders are broadly categorized into model-based and model-free approaches. Model-based decoders augment conventional algorithms, such as Belief Propagation (BP)(Richardson and Urbanke 2002) and Min-Sum (MS)(Fossorier, Mihaljevic, and Imai 1999), by using neural networks to learn the message-passing process. This paradigm has been extensively explored across various architectures and code types (Dai et al. 2021; Kwak et al. 2022, 2023; Lugosch and Gross 2017; Nachmani and Wolf 2019, 2021; Marinkovic et al. 2010; Zhu et al. 2020),

consistently achieving superior performance over conventional algorithms (Matsumine and Ochiai 2024). However, they are often limited by challenges in capturing long-range dependencies and the reliance on the underlying decoding algorithm.

In contrast, model-free decoders treat decoding as a learning problem without depending on problem-specific algorithms. While early fully-connected architectures (Gruber et al. 2017; Cammerer et al. 2017; Kim et al. 2018) struggled with overfitting, (Bennatan, Choukroun, and Kisilev 2018) proposes the pre-processing by decomposing the magnitude and syndrome vector to address overfitting issues for model-free decoders. Inspired by the recent breakthrough of Transformers (Vaswani et al. 2017), ECCT (Choukroun and Wolf 2022b) pioneered by applying self-attention to the channel output, and modeling decoding as an auto-regressive sequence-to-sequence task. Several works have improved based on this approach: FECCT (Choukroun and Wolf 2024a) improved generalization, DC-ECCT (Choukroun and Wolf 2024b) enabled joint encoder-decoder training, and CrossMPT (Park et al. 2025) introduced cross-attention to achieve better performance and efficiency among auto-regressive methods. Recently, diffusion generative models have emerged as a powerful alternative. DDECC (Choukroun and Wolf 2023) frames decoding as a denoising process, modeling the AWGN channel as the forward diffusion step, and offers performance gains over auto-regressive decoders.

**Diffusion Generative Models.** Diffusion generative models (Sohl-Dickstein et al. 2015; Ho, Jain, and Abbeel 2020; Song et al. 2020; Karras et al. 2022) learn to reverse a forward noising process by estimating the data's score function (Stein 1972). They have achieved state-of-the-art performance in synthesizing high-fidelity samples across diverse domains like images (Dhariwal and Nichol 2021; Rombach et al. 2022; Podell et al. 2024), video (He et al. 2022; Blattmann et al. 2023), text generation (Lou, Meng, and Ermon 2023; Nie et al. 2025), and graphs (Sun and Yang 2023; Li et al. 2023; Lei et al. 2025). However, a primary limitation of diffusion models is the significant computational overhead during inference, due to the iterative nature of the denoising process. To mitigate this, numerous accelerated sampling methods have been developed (Song, Meng, and Ermon 2020; Lu et al. 2022).

This framework is particularly well-suited for Error Correction Code (ECC) decoding, where the AWGN channel naturally models the forward process (Choukroun and Wolf 2022a). While diffusion-based decoders have reached state-of-the-art performance (Choukroun and Wolf 2022a; Park et al. 2024), they inherit the same computational inefficiency compared to methods like auto-regressive decoders (Choukroun and Wolf 2022b; Park et al. 2024). To address this issue, we draw inspiration from Consistency Models (CMs) (Song et al. 2023; Song and Dhariwal 2023; Geng et al. 2024), a novel technique for accelerating diffusion models. CMs are designed to directly learn the reverse denoising trajectory, enabling mapping noisy samples to the target data distribution in one step and maintaining high generation quality.

## Consistency Sampling Algorithm

Once we obtain the well-trained consistency model $f_\theta$ following Algorithm 1, we can simply apply the one-step sampling to estimate the codeword $\hat{\mathbf{x}}_0$, given any received signals $y$.

Algorithm 2: Error Correction Consistency One-step Sampling

**Require:** Consistency Model $f_\theta$, parity-check matrix $\mathbf{H}$.
**for** Test batch noisy signals $y$ **do**
  $e_t^\dagger = \mathcal{L}_{\text{Soft-syn}}(y, H)$ ▷ Calculate soft-syndrome
  $\hat{\mathbf{x}}_0 = f_\theta(y, e_t^\dagger)$ ▷ Estimate clean codeword with one-step
**end for**

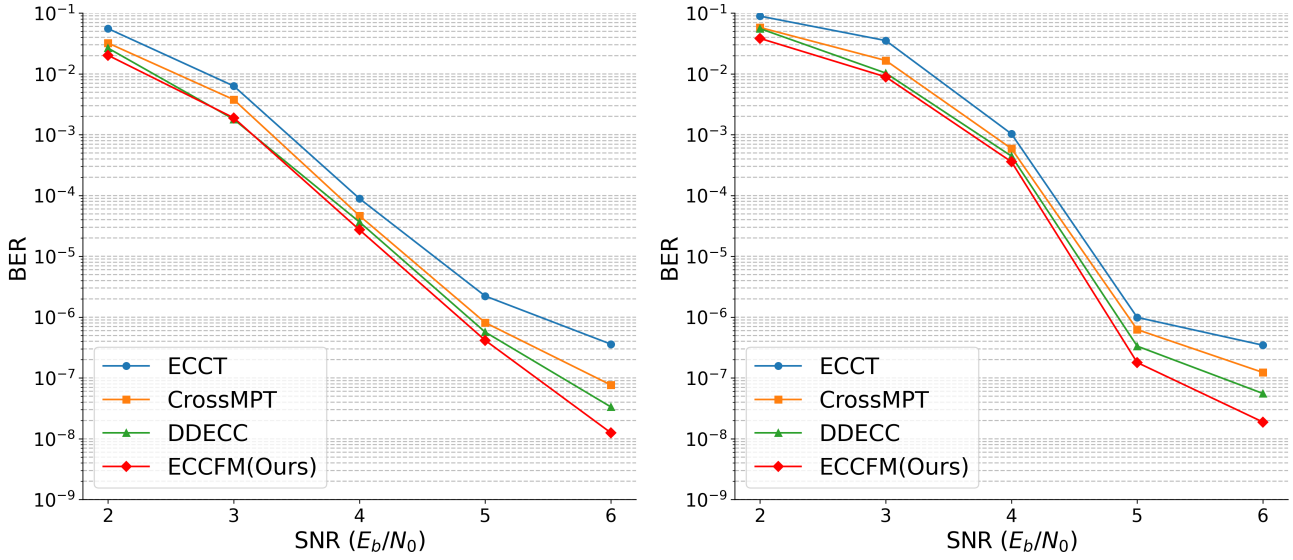
## Detailed Experimental Setting

We provide the specific design choices of ECCFM and the listed training hyperparameters. Following the design in (Choukroun and Wolf 2022b, 2023; Park et al. 2024), we generate the training set by corrupting the all-zero codeword $\mathbf{x}_0$ with AWGN noise $z$, following the diffusion process $y = \mathbf{x}_0 + \sqrt{\bar{\beta}_N} \cdot \epsilon$, where $\epsilon \sim \mathcal{N}(0, I)$. The diffusion process was configured with a total of $N = n - k + 5$ steps. We employed a linear variance schedule with $\beta_i = 0.01$ for short codes and a more fine-grained $\beta_i = 0.0025$ for medium-to-long codes. Each epoch consisted of 1,000 steps with a minibatch size of 128. To ensure a fair comparison, all neural models were implemented with a fixed architecture ($N = 6$ layers, $d = 128$ hidden dimensions). Each test task was run until at least 500 error codes were observed, under a maximum of $10^7$ test instances.

## Additional experimental results

### Additional Results on Long Codes

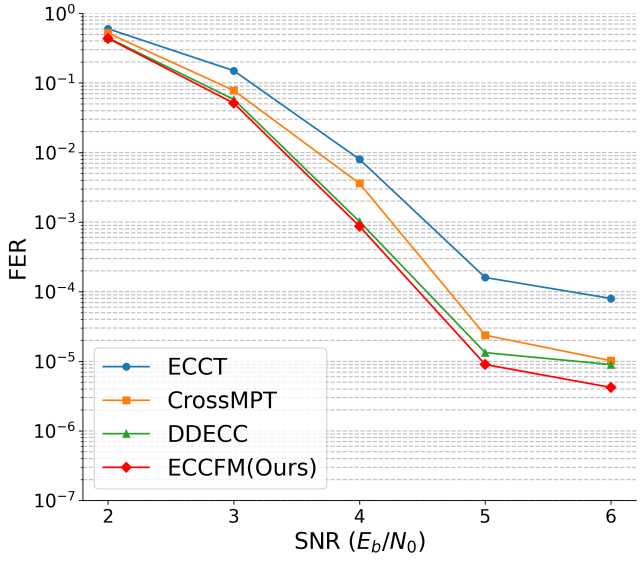
As established in Figure 4, ECCFM demonstrates scalability, achieving competitive performance on both short and medium-to-long codes. To further validate this, we present additional results for LDPC codes of varying lengths and rates, specifically LDPC($n = 204, k = 102$) and LDPC($n = 529, k = 440$). The BER and FER results in Figure 6 confirm that ECCFM improves decoding performance while maintaining its high inference speed.



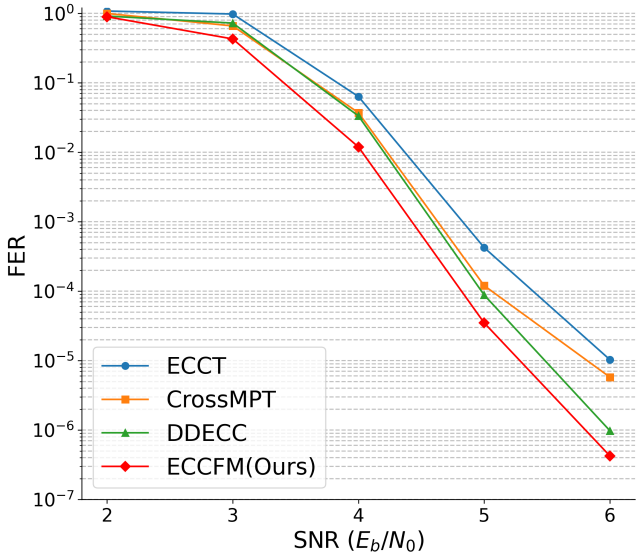

(a) BER on LDPC(204,102)




(b) BER on LDPC(529,440)




(c) FER on LDPC(204,102)




(d) FER on LDPC(529,440)

Figure 6: Performance comparison in terms of Bit Error Rate (BER) and Frame Error Rate (FER) for two LDPC codes with different blocklengths and rates: LDPC($n = 204, k = 102$) and LDPC($n = 529, k = 440$). Our method is evaluated against ECCT, CrossMPT, and DDECC.

Table 2: Training hyperparameters and design choices.

| **Parameters** | **Design Choice** |
|---|---|
| Consistency Loss | $d(\cdot,\cdot) =$ Binary Cross Entropy$(\hat{\mathbf{x}}_0, \mathbf{x}_0)$ |
| Syndrome Weight | $\lambda = 0.01$ |
| Training Epoch | 1500 |
| Mini-batch | 1000 |
| Training Batchsize | 128 |
| Test Numbers | At least 500 error cases and at most $10^7$ total numbers |
| Test Batchsize | 2048 for short codes, 256 for medium-to-long codes |
| Weighting Function | $w(t) = 1$ |
| Total Diffusion Steps | $N = n - k + 5$ |
| Forward Schedule | $\beta_i = 0.01$ for short codes, $\beta_i = 0.0025$ for medium-to-long codes |
| Time Step | $t \sim \mathcal{U}\{1, 2, \ldots, N\}$ |
| Scaling Factor | $\alpha = 0.8$ |
| Initial Learning Rate | $\eta = 1e^{-4}$ |
| Learning Rate Schedule | Cosine Decay |
| Decay Rate | $\eta' = 5e^{-7}$ |
| Exponential Moving Average Ratio | EMA$= 0.999$ |

### Additional Results on Inference Time and Throughput

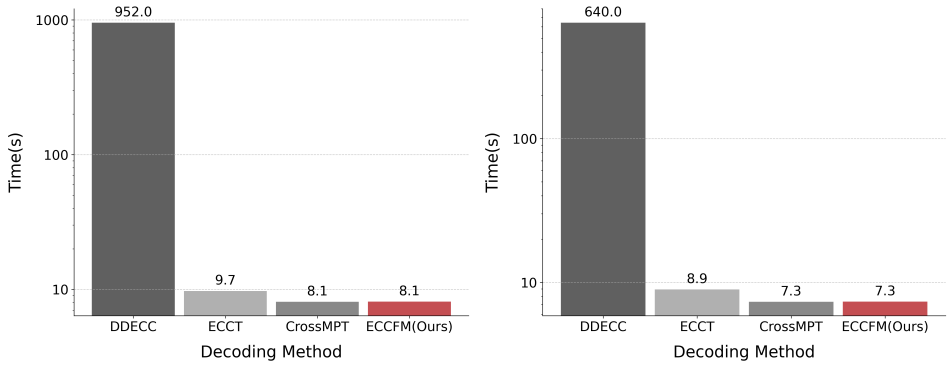
To evaluate the practical efficiency of ECCFM, we benchmarked its inference time and throughput on both POLAR and LDPC codes against established baselines (ECCT, CrossMPT, and DDECC). Inference time was measured as the total duration in seconds to decode $10^6$ samples, while throughput was defined as the number of samples decoded per second. As illustrated in Figure 5, ECCFM achieves a speedup over the denoising diffusion method, DDECC. The advantage scales with code complexity, growing from a 30x speedup on short codes to over 100x on longer codes. Further analysis, detailed in Figure 7, Figure 8, Figure 9, and Figure 10 confirms that these efficiency gains are consistent across a wide range of code types, lengths, and rates. This performance demonstrates that ECCFM provides a significant improvement in decoding speed, particularly for long codes where latency is a critical bottleneck.

### Ablation Study: Model-Agnostic Property of ECCFM

As stated in Numerical Results, ECCFM operates as a model-agnostic training framework, delivering performance gains that are independent of the model architecture (e.g., GNN, Transformer, or Cross-Attention Transformer). To validate this claim, we conducted an ablation study. We isolated our training methodology by applying it to the architecture of a different baseline, ECCT. As detailed in Table 3, we trained the ECCT backbone using the ECCFM objective and compared its performance to the original ECCT model. The results demonstrate a performance improvement, confirming that our method is adaptable to different backbones and that the observed gains are attributable to the training framework itself, not the specific model architecture.

### Analysis on Computational Overhead of iterative denoising phase

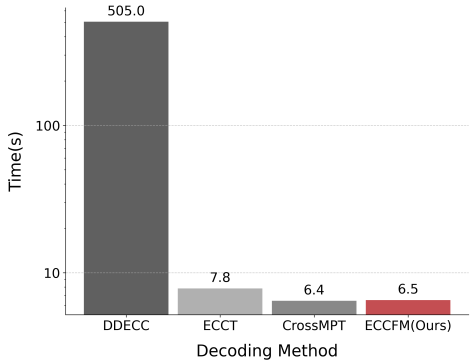
To point out the ECCFM's speed advantage, we analyzed the iterative convergence of the denoising diffusion framework. Specifically, we measured the average number of inference steps required for the DDECC model to converge to a valid codeword (i.e., achieve a zero syndrome, $e_t = 0$). As detailed in Table 4, the computational overhead for DDECC increases substantially under these three conditions: longer codes, lower code rates, and lower SNRs. This is precisely the bottleneck that ECCFM's one-step decoding offers a consistent gain in efficiency, particularly in these difficult decoding scenarios.



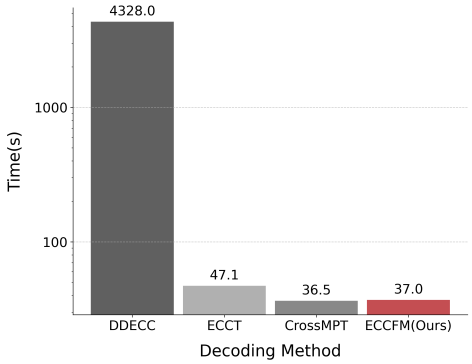

(a) LDPC(121,60)




(b) LDPC(121,70)

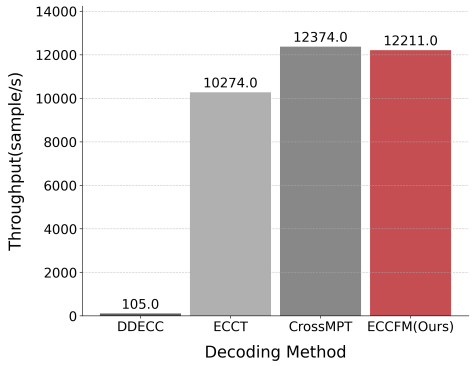



(c) LDPC(121,80)

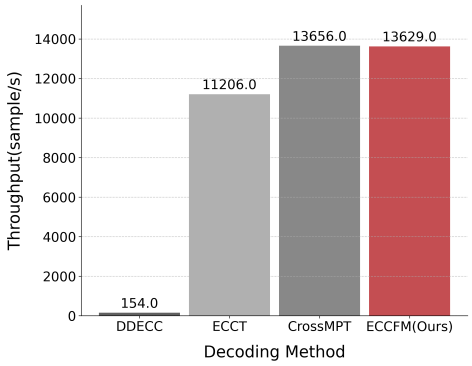



(d) LDPC(529,440)

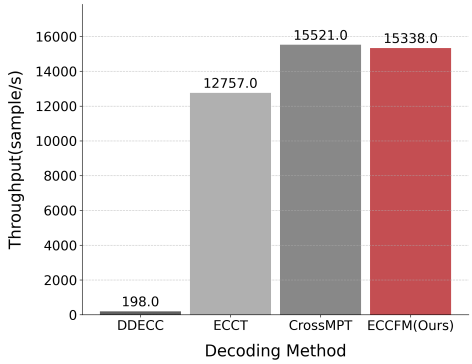
Figure 7: Inference time on LDPC($n = 121, k = 60$), LDPC($n = 121, k = 70$), LDPC($n = 121, k = 80$) and LDPC($n = 529, k = 440$), comparing with ECCT, CrossMPT and DDECC.



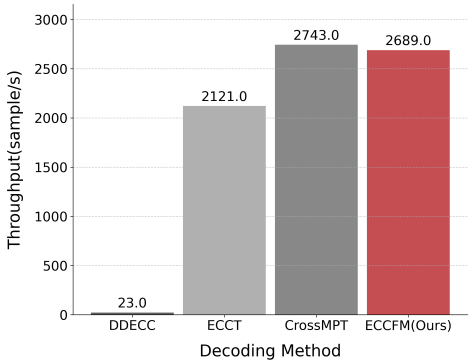

(a) LDPC(121,60)




(b) LDPC(121,70)

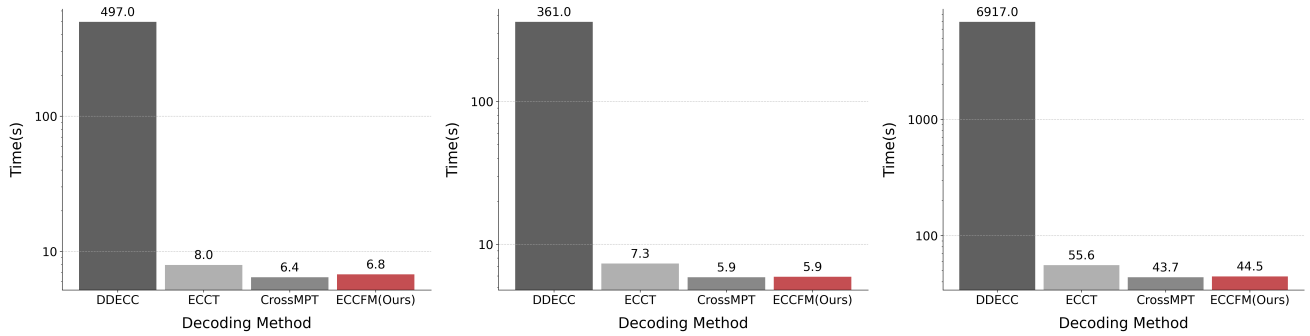



(c) LDPC(121,80)



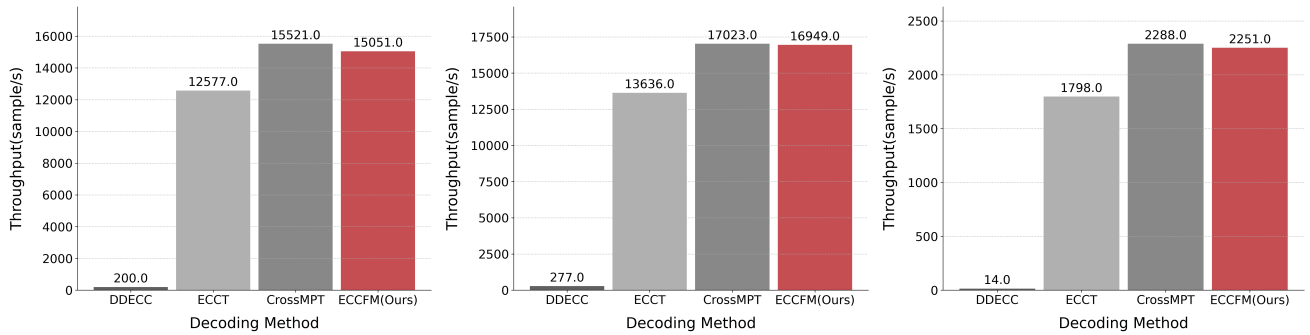

(d) LDPC(529,440)

Figure 8: Throughput on LDPC($n = 121, k = 60$), LDPC($n = 121, k = 70$), LDPC($n = 121, k = 80$) and LDPC($n = 529, k = 440$).




(a) POLAR(128,86) (b) POLAR(128,96) (c) POLAR(512,384)

Figure 9: Inference time on POLAR($n = 128, k = 86$), POLAR($n = 128, k = 96$) and POLAR($n = 512, k = 384$), comparing with ECCT, CrossMPT and DDECC.




(a) POLAR(128,86) (b) POLAR(128,96) (c) POLAR(512,384)

Figure 10: Inference time on POLAR($n = 128, k = 86$), POLAR($n = 128, k = 96$) and POLAR($n = 512, k = 384$).

Table 3: Performance comparison of ECCFM versus the standard ECCT, using an identical ECCT backbone on POLAR and LDPC codes.

| **Architecture** | | **ECCT Backbone** | | | | | |
|---|---|---|---|---|---|---|---|
| **Code Type** | **Parameters** | ECCT | | | **ECCFM(ECCT)** | | |
| | | **4** | **5** | **6** | **4** | **5** | **6** |
| **POLAR** | (64,32) | 6.87 | 9.21 | 12.15 | **7.12** | **9.77** | **12.71** |
| | (64,48) | 6.21 | 8.31 | 10.85 | **6.38** | **8.55** | **11.23** |
| | (128,64) | 5.79 | 8.45 | 11.10 | **7.32** | **11.03** | **14.87** |
| | (128,86) | 6.29 | 8.98 | 12.82 | **7.18** | **10.17** | **15.02** |
| | (128,96) | 6.30 | 9.04 | 12.40 | **6.86** | **9.94** | **13.83** |
| **LDPC** | (121,60) | 5.12 | 8.21 | 12.80 | **5.55** | **8.86** | **13.97** |
| | (121,70) | 6.30 | 10.11 | 15.50 | **6.87** | **11.21** | **16.13** |
| | (121,80) | 7.27 | 11.21 | 17.02 | **7.80** | **12.03** | **17.95** |

Table 4: Convergence steps to $e_t = 0$ of the DDECC decoder on longer codes across different Signal-to-Noise Ratios ($E_b/N_0$). The results are reported in terms of the average steps(variance).

| **Code Type** | **Parameters** | **Converge Steps: Average(Variance)** | | | | |
|---|---|---|---|---|---|---|
| | | **2** | **3** | **4** | **5** | **6** |
| **POLAR** | (512,384) | 123.40(11.38) | 91.34(21.68) | 60.24(18.00) | 41.40(16.13) | 24.99(14.82) |
| **LDPC** | (204,102) | 57.10(21.51) | 39.37(10.50) | 29.47(7.50) | 21.25(6.90) | 14.24(6.31) |
| | (408,204) | 112.12(39.47) | 76.91(13.18) | 58.39(10.38) | 42.39(9.62) | 28.99(8.81) |
| | (529,440) | 89.54(8.07) | 59.25(27.75) | 27.48(9.78) | 17.06(6.95) | 9.22(6.03) |
| **WRAN** | (384,320) | 59.66(9.79) | 37.59(18.88) | 18.57(7.99) | 11.23(5.49) | 5.91(4.64) |