BELIEFFLOW: CONTINUOUS-TIME BP-STRUCTURED NEURAL DECODING FOR LDPC CODES

Anonymous authors
Paper under double-blind review

ABSTRACT

Belief propagation (BP) decodes low-density parity-check (LDPC) codes through sparse local message passing. Recent Transformer-based decoders achieve strong performance using parity-aware attention without explicitly implementing BP's update rules. This raises the question of whether strong performance can be achieved while retaining BP's computational structure. We introduce BeliefFlow, which evolves vector-valued Tanner-edge states in continuous time using a shared neural vector field. Its local operators combine attention-based aggregation over destination-excluded neighborhoods with gated, element-wise box-plus operations. Training uses duration-aware loss reweighting to emphasize difficult examples, while randomized perturbations generate diverse retry trajectories at inference. Across six LDPC codes, the 0.34M-parameter decoder leads the comparison in 17 of 18 code/SNR settings, based on reported $-\ln(\mathrm{BER})$ scores or lower bounds. These results show that expressive local updates and perturbation-based retries can deliver strong performance within a BP-structured architecture.

1 INTRODUCTION

Low-density parity-check (LDPC) codes combine sparse parity constraints with scalable Tanner-graph inference. Belief propagation (BP) exploits this structure and provides strong decoding performance at moderate complexity (Gallager, 1962; Kschischang et al., 2001; Richardson et al., 2001). For finite-length codes, however, cycles make incoming messages statistically dependent, and a finite iteration budget may end before loopy BP reaches a valid codeword.

Neural BP addresses these failures by unrolling BP into a differentiable network and learning weights or corrections (Nachmani et al., 2016; 2018). Later work introduced neural min-sum rules, hypernetworks, and graph neural decoders while retaining an explicit message-passing graph (Nachmani & Wolf, 2019; Kwak et al., 2023; Cammerer et al., 2022).

Recently, Transformer (Vaswani et al., 2017) decoders have reported strong performance by using the parity-check matrix mainly as a mask or conditioning signal while replacing BP updates with generic learned interactions (Choukroun & Wolf, 2022; 2024; Park et al., 2025; Lei et al., 2025; Helvits & Nachmani, 2026).

This empirical gap raises a question: *does BP's computational structure limit learned decoding, or can richer local operators improve its performance?* A single model cannot settle this question universally. It can, however, test whether a competitive decoder must abandon BP's computational skeleton. We approach this test with **BeliefFlow**, which retains directed Tanner-edge states, destination-excluded aggregation neighborhoods, and a BP-style check-neighborhood interaction while learning richer vector-valued local updates.

BeliefFlow represents each check-to-variable state by a vector and models its evolution with an ordinary differential equation (ODE). The learned vector field first forms variable-to-check features by attention-weighted aggregation over destination-excluded check neighbors. It then applies a gated, element-wise aggregation over destination-excluded variable neighbors. A channel-conditioned readout maps the terminal edge state to bit logits. The construction retains the BP graph prior while replacing conventional scalar updates with learned vector operators.

The main contributions are:

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.

- A neural vector field is formulated on directed Tanner edges, with vector-valued messages and learned aggregation over BP's destination-excluded neighborhoods.
- A duration-aware loss reweighting scheme is proposed to emphasize difficult samples based on the cumulative time during which the parity-and-confidence test is not satisfied.
- A perturbation-based retry strategy is proposed using random node-feature assignments to generate diverse decoding trajectories without persistent Tanner-node identities.
- Across six LDPC codes, we evaluate decoding accuracy, solver choice, integration time, component ablations, and cross-code transfer against conventional and neural baselines.

Experiments suggest that the 0.34M-parameter BeliefFlow decoder leads the comparison in 17 of 18 code/SNR settings by reported $-\ln(\mathrm{BER})$ scores or lower bounds. Zero-shot experiments further show that the learned local updates can transfer across code lengths and rates without retraining, with performance varying across source-target code pairs.

2 Related work

2.1 BP-based LDPC decoding

LDPC decoding is built on sparse Tanner-graph inference (Gallager, 1962; Tanner, 1981; Kschischang et al., 2001). Classical refinements preserve scalar log-likelihood messages while modifying the update law or its dynamics: min-sum and its normalized or offset corrections trade complexity against approximation error (Fossorier et al., 1999; Chen & Fossorier, 2002; Chen et al., 2005), whereas layered, serial, and residual schedules and successive relaxation accelerate or stabilize convergence (Hocevar, 2004; Sharon et al., 2007; Vila Casado et al., 2010; Hemati & Banihashemi, 2006). These methods improve a hand-specified BP/MS map rather than learning the representation carried by each Tanner edge.

Continuous-time formulations also predate neural ODEs. Analog sum-product circuits were studied by Loeliger et al. (2001); Hemati & Banihashemi (2006) expressed LDPC BP/MS as first-order nonlinear differential equations and related Euler discretization to successive relaxation; and Mostafa et al. (2016) mapped LDPC BP to an untrained continuous-time recurrent neural network whose connectivity is fixed by the parity-check matrix. Gradient-flow decoding instead evolves a codeword-domain state under a parity-aware potential (Wadayama & Wei, 2025). In this work, BeliefFlow learns continuous-time dynamics for vector-valued Tanner-edge messages, with element-wise box-plus interactions over check-node neighborhoods.

2.2 Learning-based channel decoding

Model-driven neural decoders learn within an iterative message-passing graph. Weighted and recurrent BP learn edge or iteration corrections, including parameter sharing and successive relaxation (Nachmani et al., 2016; 2018); neural min-sum, hypernetwork, pruning, and error-floor-oriented variants further improve accuracy or implementation cost (Lugosch & Gross, 2017; Nachmani & Wolf, 2019; Buchberger et al., 2021; Kwak et al., 2023). General belief-propagation neural networks, protograph decoders, and channel-decoding GNNs learn shared vector operators across graph locations or codes (Kuck et al., 2020; Dai et al., 2021; Cammerer et al., 2022; Tian et al., 2023). These methods organize inference as a finite sequence of discrete message-passing steps. In this work, BeliefFlow combines BP-structured message passing with learned continuous-time dynamics.

A parallel line replaces explicit BP's update rules with more general code-aware interactions. Syndrome-based networks and transformer decoders such as ECCT, FECCT, CrossMPT, and EfficientMPT condition recurrent or attention operators on parity structure (Bennatan et al., 2018; Choukroun & Wolf, 2022; 2024; Park et al., 2025; 2026). Diffusion and flow decoders instead treat decoding as codeword-space denoising: DDECC performs iterative reverse diffusion, SyndromeFlow (ECCFM) learns a soft-syndrome-parameterized one-step consistency map, and SB-ECC follows a parity-guided score-based probability-flow ODE (Choukroun & Wolf, 2023; Lei et al., 2025; Helvits & Nachmani, 2026). These methods use parity-check information to guide learned decoding through syndrome features or parity-aware attention mechanisms, but do not explicitly implement BP's message-update rules.

3 BACKGROUND

3.1 PROBLEM STATEMENT

Let $\mathscr{C}$ be a binary linear (n, k) code with parity-check matrix $H \in \{0,1\}^{m\times n}$. A message $\boldsymbol{u} \in \{0,1\}^k$ is encoded as a codeword $\boldsymbol{x} \in \mathscr{C}$ satisfying $H\boldsymbol{x} = \mathbf{0}$ over $\mathrm{GF}(2)$. Binary phase-shift keying (BPSK) maps each coded bit to $s_v = 1 - 2x_v \in \{+1, -1\}$. Transmission over an additive white Gaussian noise (AWGN) channel gives

$$\boldsymbol{y} = \boldsymbol{s} + \boldsymbol{w}, \qquad \boldsymbol{w} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I_n). \tag{1}$$

When σ^2 is known, the channel log-likelihood ratio (LLR) is

$$L_v = \log \frac{p(y_v \mid x_v = 0)}{p(y_v \mid x_v = 1)} = \frac{2y_v}{\sigma^2}. \tag{2}$$

A decoder maps $\boldsymbol{y}$ to bit estimates $\hat{\boldsymbol{x}} \in \{0,1\}^n$. We say that a predicted codeword $\hat{\boldsymbol{x}}$ is parity-valid if it satisfies $H\hat{\boldsymbol{x}} = \mathbf{0}$ over $\mathrm{GF}(2)$, while it may not be the same as the true codeword.

3.2 BELIEF PROPAGATION AND ITS CONTINUOUS-TIME ANALOG

The Tanner graph $\mathcal{G} = (\mathcal{V}, \mathcal{C}, \mathcal{E})$ is bipartite, with one variable node $v \in \mathcal{V}$ per column of H, one check node $c \in \mathcal{C}$ per row, and $(c, v) \in \mathcal{E}$ if and only if $H_{cv} = 1$. We write $\mathcal{N}(v)$ and $\mathcal{N}(c)$ for the corresponding neighbor sets. In flooding BP, check-to-variable messages are initialized to zero and the two extrinsic updates are

$$m_{v\to c}^{(\ell+1)} = L_v + \textstyle\sum_{c'\in\mathcal{N}(v)\setminus\{c\}} m_{c'\to v}^{(\ell)}, \quad m_{c\to v}^{(\ell+1)} = 2\,\mathrm{atanh}\Big[\prod_{v'\in\mathcal{N}(c)\setminus\{v\}} \tanh\Big(m_{v'\to c}^{(\ell+1)}/2\Big)\Big]. \tag{3}$$

After an iteration, the posterior LLR and hard decision are given by

$$L_v^{\mathrm{post}} = L_v + \textstyle\sum_{c\in\mathcal{N}(v)} m_{c\to v}, \qquad \hat{x}_v = \mathbb{1}[L_v^{\mathrm{post}} < 0]\,. \tag{4}$$

Analog implementations of sum-product and MAP decoding were studied before neural ODEs (Hagenauer et al., 1999; Loeliger et al., 2001; Moerz et al., 2000). Hemati & Banihashemi (2006) and Xiao & Banihashemi (2009) subsequently described asynchronous LDPC decoding by first-order nonlinear dynamics and extended the associated relaxation analysis. In the LLR domain, a representative continuous relaxation is

$$\begin{aligned}
\tau_{v\to c}\tfrac{\mathrm{d}m_{v\to c}(t)}{\mathrm{d}t} &= \Big[L_v + \textstyle\sum_{c'\in\mathcal{N}(v)\setminus\{c\}} m_{c'\to v}(t)\Big] - m_{v\to c}(t),\\
\tau_{c\to v}\tfrac{\mathrm{d}m_{c\to v}(t)}{\mathrm{d}t} &= 2\,\mathrm{atanh}\Big[\prod_{v'\in\mathcal{N}(c)\setminus\{v\}} \tanh(m_{v'\to c}(t)/2)\Big] - m_{c\to v}(t),
\end{aligned} \tag{5}$$

where the τ terms are positive time constants.

4 BELIEFFLOW: CONTINUOUS-TIME NEURAL DYNAMICS ON TANNER EDGES

BeliefFlow replaces each scalar check-to-variable message by a vector-valued Tanner-edge state. Stacking directed states in a fixed edge order gives

$$\boldsymbol{M}_{\mathcal{C}\to\mathcal{V}}(t) := \big[\boldsymbol{m}_{c\to v}(t)^{\mathsf{T}}\big]_{(c,v)\in\mathcal{E}} \in \mathbb{R}^{|\mathcal{E}|\times d_m}. \tag{6}$$

The raw channel observations $\boldsymbol{y} := [y_v]_{v\in\mathcal{V}} \in \mathbb{R}^{n\times 1}$ are embedded independently at the variable nodes by a multi-layer perceptron (MLP)

$$\widetilde{\boldsymbol{Y}} := \big[\widetilde{\boldsymbol{y}}_v^{\mathsf{T}}\big]_{v\in\mathcal{V}} = \mathrm{MLP}_{\mathrm{map}}(\boldsymbol{y}) \in \mathbb{R}^{n\times d_m}. \tag{7}$$

The dynamics are the following ODE:

$$\begin{cases} \frac{\mathrm{d}\boldsymbol{M}_{\mathcal{C}\to\mathcal{V}}(t)}{\mathrm{d}t} = f_\theta(\boldsymbol{M}_{\mathcal{C}\to\mathcal{V}}(t); \widetilde{\boldsymbol{Y}}) - \boldsymbol{M}_{\mathcal{C}\to\mathcal{V}}(t), \\ \boldsymbol{M}_{\mathcal{C}\to\mathcal{V}}(0) = \mathbf{0}, \quad t \in (0, T]. \end{cases} \tag{8}$$

Here f_θ is the shared neural update target defined in Section 4.1; the ODE vector field is $f_\theta(\boldsymbol{M}; \widetilde{\boldsymbol{Y}}) - \boldsymbol{M}$. For each decoding attempt, the graph and random feature assignment are fixed and omitted from the notation. The terminal state is gathered at each variable node and demapped to an LLR-like logit,

$$\hat{L}_v(T) = \mathrm{MLP}_{\mathrm{demap}}\left(\tilde{\boldsymbol{y}}_v + \textstyle\sum_{c \in \mathcal{N}(v)} \boldsymbol{m}_{c \to v}(T)\right). \tag{9}$$

The same readout applies at any time t. Positive $\hat{L}_v(t)$ favors bit zero, and $\hat{x}_v(t) = \mathbb{1}[\hat{L}_v(t) < 0]$.

Applying forward Euler with a fixed step size Δt to Equation (8) yields

$$\boldsymbol{M}_{\mathcal{C}\to\mathcal{V}}^{(\ell+1)} = (1 - \Delta t)\boldsymbol{M}_{\mathcal{C}\to\mathcal{V}}^{(\ell)} + \Delta t f_\theta(\boldsymbol{M}_{\mathcal{C}\to\mathcal{V}}^{(\ell)}; \widetilde{\boldsymbol{Y}}). \tag{10}$$

For $0 < \Delta t < 1$, this corresponds to a relaxed residual update toward the learned BP-structured update target. The continuous-time formulation defines the learned message dynamics independently of a particular numerical discretization (Chen et al., 2018). This allows the integration horizon, step size, and solver to be varied while keeping the learned parameters fixed. The cumulative time spent failing the parity-and-confidence test can also serve as a proxy for decoding difficulty, enabling loss reweighting to emphasize harder training examples.

4.1 BP-STRUCTURED NEURAL MESSAGE-PROPAGATION FUNCTION

The BP-structured message-propagation function $f_\theta(\boldsymbol{M}_{\mathcal{C}\to\mathcal{V}}(t); \widetilde{\boldsymbol{Y}})$ is illustrated alongside standard BP in Figure 1. It retains BP's directed edge states and destination-excluded aggregation neighborhoods, while replacing scalar updates with attention-based vector aggregation and gated box-plus interactions.

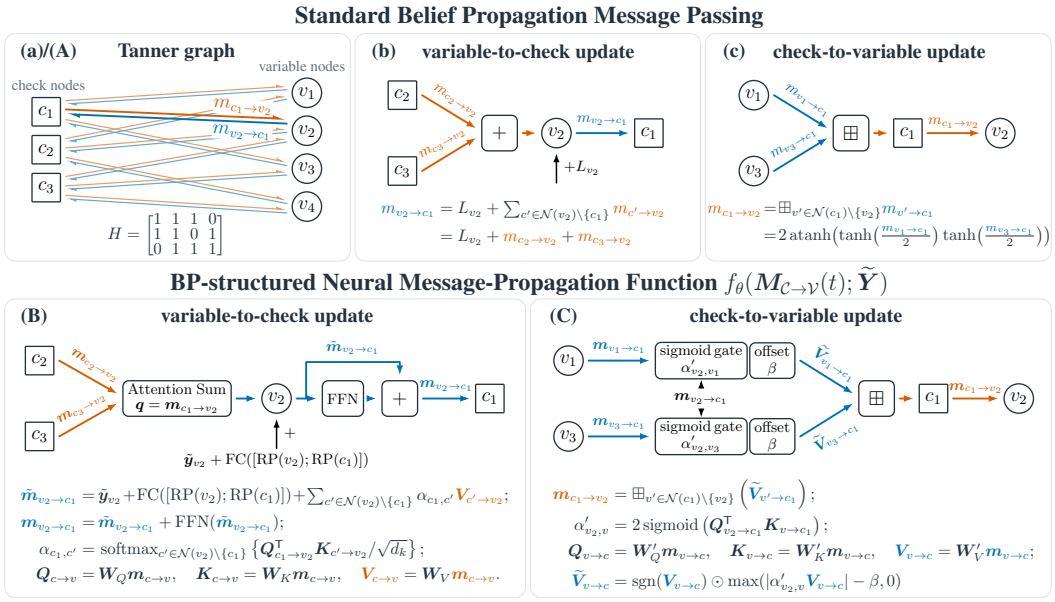

Figure 1: Standard BP (top) and BeliefFlow's neural update target $f_\theta(\boldsymbol{M}_{\mathcal{C}\to\mathcal{V}}; \widetilde{\boldsymbol{Y}})$ (bottom). Both stages aggregate over destination-excluded neighborhoods, with attention and gates conditioned on the destination edge. RP denotes random node-feature assignments from either frozen or learnable banks. The bottom-right output is an update target, rather than the integrated state at the next step.

For an edge $e = (c, v)$, the aggregation neighborhoods exclude the destination incidence:

$$\mathcal{E}_v(e) = \{(c', v) \in \mathcal{E} : c' \neq c\}, \qquad \mathcal{E}_c(e) = \{(c, v') \in \mathcal{E} : v' \neq v\}. \tag{11}$$

The variable-to-check stage replaces the unweighted sum in Equation (3) with sparse multi-head attention (Vaswani et al., 2017). For destination (c, v), the query comes from $\boldsymbol{m}_{c\to v}$; keys and values come from $\mathcal{E}_v(e)$. The aggregated values are added to the channel embedding and an edge feature derived from the node features at both endpoints. A residual feed-forward network (FFN) then produces the intermediate variable-to-check feature $\boldsymbol{m}_{v\to c}$.

The check-to-variable stage aggregates transformed variable-to-check features over $\mathcal{E}_c(e)$. Each input receives an independent sigmoid gate conditioned on the destination query. Unlike softmax weights, these gates do not impose a sum-to-one constraint. The gated inputs are combined element-wise using box-plus operator

$$x \boxplus y = 2\,\mathrm{atanh}[\tanh(x/2)\tanh(y/2)] = \mathrm{sgn}(x)\,\mathrm{sgn}(y)\min(|x|,|y|) + \log\tfrac{1+e^{-|x+y|}}{1+e^{-|x-y|}}. \quad (12)$$

After gating, a learnable scalar offset β is subtracted from the absolute value of each component, with negative results clipped to zero, as shown in Figure 1. Normalization and numerical implementation details are provided in Section A.4. This construction retains a BP-structured message propagation mechanism while allowing learned feature transformations.

4.2 Random feature perturbations

Loops in finite-length Tanner graphs can induce correlated BP messages (Kschischang et al., 2001; Xiao & Banihashemi, 2004). BeliefFlow uses random feature perturbations (RP) to vary its learned dynamics across decoding attempts. Separate variable-node and check-node feature banks are initialized with independent standard-normal entries. During both training and inference, feature vectors are sampled without replacement from the respective banks for each sample and decoding attempt. The resulting node-feature assignments remain fixed throughout integration. A linear projection maps the concatenated endpoint features to an edge contribution added alongside $\tilde{\boldsymbol{y}}_v$. We evaluate frozen and learnable variants of this perturbation mapping. The frozen variant retains the initial bank and projection parameters, whereas the learnable variant optimizes both jointly with the decoder. At inference, both variants keep these parameters fixed and resample only the node-feature assignments. The frozen variant is the default, motivated by its generally stronger zero-shot cross-code performance in Section 6.4.

The single-attempt ablation does not show a consistent benefit from adding a feature bank alone. Both frozen and learnable banks improve the reported results when combined with retries. We therefore interpret the perturbations primarily as a source of retry diversity, rather than evidence of learned cycle suppression. Perturbation-based BP decoders also vary decoding trajectories to mitigate failures (Varnica et al., 2007; Leduc-Primeau et al., 2012; Lee et al., 2020). Here, feature reassignment perturbs the neural update target without changing the channel observation or model parameters. The retry ablation in Section 6.3 evaluates this combined inference mechanism.

5 Training and decoding procedures

5.1 Capped-horizon training and trajectory statistics

We integrate BeliefFlow with the differentiable solvers in `torchdiffeq` (Chen et al., 2018; 2021). Let T denote the maximum integration horizon and let $\boldsymbol{M}_i(t)$ denote sample i's message state. Using the readout in Equation (9) at time t, define the acceptance indicator

$$A_i(t) = \mathbb{1}\left[\min_v |\hat{L}_{iv}(t)| > \gamma_{\mathrm{conf}} \text{ and } H\hat{\boldsymbol{x}}_i(t) = \mathbf{0}\right], \qquad \gamma_{\mathrm{conf}} = \log\tfrac{0.5+\delta}{0.5-\delta}, \quad (13)$$

where $0 < \delta < 0.5$ specifies the minimum required deviation of the sigmoid outputs from 0.5.

During training, the trajectory and its accumulated statistics satisfy

$$\begin{cases} \frac{\mathrm{d}\boldsymbol{M}_i(t)}{\mathrm{d}t} = f_\theta(\boldsymbol{M}_i(t); \widetilde{\boldsymbol{Y}}_i) - \boldsymbol{M}_i(t), \\ \frac{\mathrm{d}D_i(t)}{\mathrm{d}t} = 1 - A_i(t), \quad \frac{\mathrm{d}E_i(t)}{\mathrm{d}t} = \frac{1}{|\mathcal{E}|d_m}\left\|\frac{\mathrm{d}\boldsymbol{M}_i(t)}{\mathrm{d}t}\right\|_F^2, \qquad 0 \le t < T. \\ \boldsymbol{M}_i(0) = \mathbf{0}, \quad D_i(0) = 0, \quad E_i(0) = 0, \end{cases} \quad (14)$$

At the training horizon T, the accumulated statistics are

$$D_i(T) = \int_0^T [1 - A_i(t)]\,\mathrm{d}t, \qquad E_i(T) = \frac{1}{|\mathcal{E}|d_m}\int_0^T \left\|\frac{\mathrm{d}\boldsymbol{M}_i(t)}{\mathrm{d}t}\right\|_F^2 \mathrm{d}t. \quad (15)$$

5.2 DURATION-AWARE LOSS REWEIGHTING

Easy received words can dominate an unweighted training objective at high SNR. The accumulated duration $D_i(T)$ serves as a difficulty proxy, with the following override for incorrectly decoded training words:

$$d_i = \begin{cases} D_i(T), & \hat{\boldsymbol{x}}_i(T) = \boldsymbol{x}_i, \\ T, & \hat{\boldsymbol{x}}_i(T) \neq \boldsymbol{x}_i. \end{cases} \tag{16}$$

For a batch of size B, the exponential moving average is recorded

$$\bar{d}_k = \rho \bar{d}_{k-1} + (1-\rho)\tfrac{1}{B}\textstyle\sum_{i=1}^{B} d_i, \qquad 0 \le \rho < 1. \tag{17}$$

Because positive $\hat{L}$ favors bit zero, the bit-one logit supplied to binary cross-entropy is $-\hat{L}$. Writing $\mathrm{BCE}_{\mathrm{logits}}$ for binary cross-entropy averaged over bits, the training objective is

$$\mathcal{L} = \tfrac{1}{B}\textstyle\sum_{i=1}^{B}\left[\frac{d_i}{\bar{d}_k+\epsilon}\,\mathrm{BCE}_{\mathrm{logits}}\Big(-\hat{\boldsymbol{L}}_i(T), \boldsymbol{x}_i\Big) + \lambda\frac{E_i(T)}{T}\right], \tag{18}$$

where the first term reweights the prediction loss and the second penalizes the mean squared message velocity over the training horizon (Finlay et al., 2020).

5.3 RETRY-BASED DECODING

At inference, each trajectory is frozen once its readout satisfies the parity-and-confidence test in Equation (13). If the initial attempt times out, the decoder launches N_{retry} candidates in parallel. Each candidate starts from zero messages with a newly sampled assignment of bank features. Shared integration stops when at least K candidates are accepted or the maximum horizon is reached. The final prediction is obtained by bitwise majority voting over all accepted retry candidates. If none are accepted, all retry candidates vote using their predictions at the integration horizon.

6 EXPERIMENTS

6.1 EXPERIMENTAL SETUP

The main experiments cover the six LDPC codes listed in Table 1. Uniformly sampled information vectors are encoded, BPSK-modulated, and transmitted over AWGN channels. To prevent identical messages from appearing in both partitions of training and testing sets, each message is assigned to one of seven buckets using its deterministic hash modulo 7. Bucket 0 is used for testing and the remaining buckets for training. Preliminary results on longer codes are provided in Section A.2. Source code and pretrained models are at `anonymous.4open.science/r/BeliefFlow`.

The default hyperparameters are set as follows: dimension of the message features $d_m = 128$, number of attention heads $h = 8$, and the kinetic-energy regularization weight $\lambda = 1$ from Equation (18). Each random feature bank contains $N_{\mathrm{bank}} = 128$ entries, supporting feature assignments for the code sizes evaluated in Section 6.4.

Training. All training trajectories are integrated with RK4 up to $T_{\mathrm{train}} = 5$ using a step size of $\Delta t = 0.1$, corresponding to 50 integration steps. Samples are generated over AWGN channels with SNRs sampled uniformly from $[2, 3, 4, 5, 6, 7]$ dB. Samples whose preliminary hard decisions pass the parity checks are excluded from training. Training uses 800 epochs, each with 100×256 samples.

Testing. Default testing uses a maximum horizon $T_{\mathrm{test}} = 20$. It uses RK4 solver with step size $\Delta t = 0.1$, giving at most 200 integration steps per attempt. After a timed-out initial attempt, BeliefFlow evaluates at most $N_{\mathrm{retry}} = 256$ random feature assignments, with candidate-count stopping threshold $K = 20$. For most of the experimental settings, we report the negative natural logarithm of the bit error rate (BER): $-\ln(\mathrm{BER})$, where higher values indicate better decoding performance. It is calculated by collecting 100 frame errors or processing 10^8 codewords, whichever occurs first.

6.2 COMPARATIVE EXPERIMENTS

We compare BeliefFlow with flooding BP as a conventional baseline and SR-MS-LLR as a successive-relaxation approximation to classical continuous-time min-sum decoding (Hemati &

Banihashemi, 2006; Xiao & Banihashemi, 2009). AR-BP represents neural BP (Nachmani & Wolf, 2021), while SB-ECC, a score-based decoder, serves as a state-of-the-art (SOTA) neural decoding baseline (Helvits & Nachmani, 2026).

We evaluate BP and SR-MS-LLR using our implementations. BP uses at most 200 iterations with parity-based early stopping. SR-MS-LLR follows Xiao & Banihashemi (2009), with relaxation factor $\beta = 0.5$ and at most 200 flooding iterations. The AR-BP and SB-ECC entries are taken from the cited papers; pretrained weights were unavailable for our evaluation. We also trained SB-ECC using the authors' implementation with the default 6-layered Transformer blocks and evaluated it with 34 decoding steps. Its performance was comparable to or slightly worse than the published results with 10 decoding steps. We therefore retain the published SB-ECC values in the main table.

Table 1: Comparison with existing methods in terms of $-\ln(\mathrm{BER})$ at $E_b/N_0 \in \{4, 5, 6\}$ dB. Higher values indicate better decoding performance. Bold and underlined entries indicate the largest and second-largest displayed scores or bounds.

	Conventional decoders						Neural decoders								
	BP			SR-MS-LLR			AR-BP			SB-ECC			**BeliefFlow**		
Code(n, k)	4	5	6	4	5	6	4	5	6	4	5	6	4	5	6
LDPC$(49, 24)$	6.23	9.11	12.56	6.35	8.99	12.03	6.58	9.39	12.39	<u>7.74</u>	<u>10.88</u>	**14.63**	**8.11**	**11.19**	<u>14.54</u>
LDPC$(121, 60)$	5.83	9.57	14.64	6.03	9.87	15.39	5.22	8.31	13.07	<u>6.33</u>	<u>10.38</u>	<u>16.38</u>	**8.24**	**13.10**	**20.82**
LDPC$(121, 70)$	7.27	11.44	17.72	7.59	12.22	17.87	6.45	10.01	14.77	<u>7.75</u>	<u>12.69</u>	<u>19.24</u>	**9.40**	**14.24**	**19.32**
LDPC$(121, 80)$	8.13	12.86	18.23	8.54	12.88	18.82	7.22	11.03	15.90	<u>8.84</u>	<u>14.02</u>	<u>20.42</u>	**10.03**	**15.19**	$\geq$**22.12**†
MacKay$(96, 48)$	8.52	11.90	15.49	8.80	12.28	15.80	7.43	10.65	14.65	<u>8.97</u>	<u>12.82</u>	<u>16.25</u>	**10.21**	**13.33**	**16.34**
CCSDS$(128, 64)$	8.28	13.23	19.17	<u>9.14</u>	<u>14.17</u>	<u>19.84</u>	7.25	10.99	16.36	8.60	13.43	19.20	**10.77**	**16.28**	$\geq$**22.18**†

† No bit errors were observed over 1.0×10^8 decoded codewords. Assuming independent bit errors, the reported value is the one-sided 95% lower confidence bound on $-\ln(\mathrm{BER})$, as derived in Section A.3.

As suggested in Table 1, BeliefFlow has reported higher scores or bounds than the listed baselines in 17 of 18 settings. At 6 dB on LDPC$(49, 24)$, its score is 14.54, slightly inferior to SB-ECC's 14.63.

6.3 Ablation study and hyperparameter tuning

6.3.1 Effect of perturbations and retry decoding

Table 2 separates feature-bank choice from retry decoding. No bank, F-bank, and L-bank use one attempt, with no bank, a frozen bank, and a learnable bank, respectively. BF-F and BF-L add retries with the frozen-bank and learnable-bank variants. BF-F is the default decoder.

Retries improve every reported BF-F/F-bank and BF-L/L-bank comparison. Adding a frozen or learnable bank without retries has mixed effects relative to No bank, with degradation in most settings. Thus, the bank's main observed benefit arises in combination with retries, rather than from a uniform single-attempt improvement. Frozen and learnable banks yield broadly comparable performance with/without retries. CCSDS$(128, 64)$ is an exception: BF-L exceeds BF-F by 0.59 and 1.17 at 4 and 5 dB, respectively. However, the cross-code comparison in Section 6.4 favors the frozen bank in most available transfer settings, motivating our default choice. The single-attempt variants already exceed the reported SB-ECC scores in several settings. These results demonstrate competitive decoding performance without retries, with retries providing further gains.

Table 2: Ablation study of random feature perturbations and retry decoding. $-\ln(\mathrm{BER})$ is reported.

	SB-ECC		No bank		F-bank		L-bank		BF-F		BF-L	
Code(n, k)	4	5	4	5	4	5	4	5	4	5	4	5
LDPC$(49, 24)$	**7.74**	**10.88**	7.33	10.25	7.39	10.32	7.29	10.08	8.11	11.19	8.05	11.01
LDPC$(121, 60)$	6.33	10.38	**6.47**	**10.70**	5.83	9.42	6.13	10.01	8.24	13.10	7.65	12.29
LDPC$(121, 70)$	**7.75**	**12.69**	7.68	12.52	7.70	12.12	7.64	12.06	9.40	14.24	9.42	14.38
LDPC$(121, 80)$	8.84	14.02	**9.32**	**14.30**	8.84	13.50	8.89	13.65	10.03	15.19	10.10	15.43
MacKay$(96, 48)$	8.97	12.82	**9.59**	12.92	9.28	12.81	9.36	**13.00**	10.21	13.33	10.20	13.37
CCSDS$(128, 64)$	8.60	13.43	**9.47**	**15.09**	8.70	13.40	9.02	13.52	10.77	16.28	11.36	17.45

6.3.2 Effect of duration-aware loss reweighting

Duration-aware loss reweighting (DLR) emphasizes difficult training words to address the imbalance between easy and difficult examples. As shown in Table 3, DLR improves the reported scores or lower bounds for five of the six codes across $E_b/N_0 \in \{4, 5, 6\}$ dB, with the largest gains at 6 dB. For example, on LDPC$(121, 60)$, the gain in $-\ln(\text{BER})$ increases from 0.44 at 4 dB to 1.93 at 6 dB. This trend is consistent with the motivation for DLR: increasing the relative contribution of difficult training examples when easy examples become more prevalent at higher SNRs. For CCSDS $(128, 64)$, DLR yields lower scores at 4 and 5 dB, while both variants record zero errors at 6 dB.

Table 3: Ablation study of DLR. Results are reported as $-\ln(\text{BER})$. Cell colors indicate relative differences between training without/with DLR.

	without DLR			with DLR		
Code(n, k)	4	5	6	4	5	6
LDPC$(49, 24)$	8.10	10.78	13.93	8.11	11.19	14.54
LDPC$(121, 60)$	7.80	12.51	18.89	8.24	13.10	20.82
LDPC$(121, 70)$	9.14	14.06	18.86	9.40	14.24	19.32
LDPC$(121, 80)$	9.95	14.58	21.42	10.03	15.19	$\geq 22.12^{\dagger}$
MacKay$(96, 48)$	10.12	12.63	15.18	10.21	13.33	16.34
CCSDS$(128, 64)$	11.18	17.20	$\geq 22.18^{\dagger}$	10.77	16.28	$\geq 22.18^{\dagger}$

† No errors were observed. 95% lower confidence bound is reported.

6.3.3 Solver sensitivity

Using the same RK4-trained weights for each code, we test the decoders with Euler, RK4 (Hairer et al., 1993), and DOPRI5 (Dormand & Prince, 1980) at $E_b/N_0 \in \{4, 5\}$ dB in Table 4. Euler achieves scores comparable to RK4 with fewer function evaluations as in Table 5, while DOPRI5 yields lower scores in several settings despite requiring more evaluations. These results demonstrate the flexibility to switch from RK4 to Euler at inference while maintaining decoding performance.

Table 5 reports the average number of function evaluations (NFE) per decoding attempt. RK4 uses four evaluations per step; DOPRI5's cost depends on adaptive step acceptance and initialization. The mean of the per-setting NFE ratios relative to Euler is approximately 3.19 for RK4 and 8.30 for DOPRI5. RK4 takes fewer steps on average, but incurs more function evaluations than Euler.

Table 4: Comparison of ODE solvers in terms of $-\ln(\text{BER})$ at $E_b/N_0 \in \{4, 5\}$ dB. Cell colors indicate relative differences to the default (RK4).

	Euler		RK4		DOPRI5	
Code(n, k)	4	5	4	5	4	5
LDPC$(49, 24)$	8.14	10.98	8.11	11.19	7.88	10.53
LDPC$(121, 60)$	7.96	12.93	8.24	13.10	8.26	12.90
LDPC$(121, 70)$	9.25	14.26	9.40	14.24	9.46	13.98
LDPC$(121, 80)$	10.02	15.03	10.03	15.19	9.71	13.88
MacKay$(96, 48)$	10.06	13.14	10.21	13.33	9.71	12.54
CCSDS$(128, 64)$	10.43	15.90	10.77	16.28	10.85	14.82

Table 5: Average number of NFE per decoding attempt at $E_b/N_0 \in \{4, 5\}$ dB. Lower values indicate fewer vector-field evaluations.

	Euler		RK4		DOPRI5	
Code(n, k)	4	5	4	5	4	5
LDPC$(49, 24)$	7.94	3.99	28.62	16.26	55.38	47.58
LDPC$(121, 60)$	20.55	4.82	55.81	16.01	72.53	54.11
LDPC$(121, 70)$	7.63	3.09	25.40	13.98	56.03	48.68
LDPC$(121, 80)$	11.42	9.07	32.24	27.97	56.20	52.27
MacKay$(96, 48)$	7.77	5.07	21.10	16.33	53.28	49.06
CCSDS$(128, 64)$	10.88	6.77	24.41	17.57	68.39	63.84

6.3.4 Integration time T and step size Δt

Table 6: Effect of integration time T at $\Delta t = 0.1$. Results are $-\ln(\text{BER})$. Cell colors indicate relative differences to the default ($T = 20$).

	$T =$ 0.5	1	2.5	5	10	**20**	40	80
LDPC$(49, 24)$	4.63	5.75	7.06	7.62	7.95	**8.11**	8.11	8.11
LDPC$(121, 60)$	4.64	6.46	7.21	7.58	7.84	**8.24**	8.44	8.67
LDPC$(121, 70)$	5.88	7.36	8.82	9.10	9.23	**9.40**	9.51	9.67
LDPC$(121, 80)$	4.36	7.32	9.43	9.86	10.04	**10.03**	10.07	10.11
MacKay$(96, 48)$	5.85	8.05	9.59	9.92	10.11	**10.21**	10.18	10.21
CCSDS$(128, 64)$	4.98	8.33	9.71	10.44	10.70	**10.77**	10.95	11.02

Table 7: Effect of step size Δt at $T = 20$. Results are $-\ln(\text{BER})$. Cell colors indicate relative differences to the default ($\Delta t = 0.1$).

	$\Delta t =$ 0.025	0.05	**0.1**	0.2	0.4	0.6	0.8	1
LDPC$(49, 24)$	8.06	8.15	**8.11**	8.12	7.98	7.73	7.26	6.90
LDPC$(121, 60)$	8.27	8.14	**8.24**	8.25	8.24	7.90	7.43	6.39
LDPC$(121, 70)$	9.44	9.40	**9.40**	9.33	9.06	8.64	8.19	7.61
LDPC$(121, 80)$	10.05	10.04	**10.03**	10.01	9.98	9.41	8.38	7.98
MacKay$(96, 48)$	10.17	10.13	**10.21**	10.12	10.16	9.73	8.84	8.39
CCSDS$(128, 64)$	10.82	10.85	**10.77**	10.78	10.18	9.05	8.41	7.53

While training uses $T_{\text{train}} = 5$ and $\Delta t = 0.1$, we evaluate different maximum integration horizons at test time while keeping the step size fixed. Table 6 reports the results for all six codes at $E_b/N_0 = 4$ dB. The scores improve rapidly as the integration horizon increases and generally approach a plateau around $T = 10$. This trend is consistent with BP-like iterative decoding, where additional message-passing rounds can improve performance before the gains saturate.

With the integration horizon fixed at $T = 20$, we evaluate decoding performance across different inference step sizes Δt. As suggested in Table 7, smaller step sizes yield performance similar to the default $\Delta t = 0.1$, while increasing Δt to 0.4 results in modest score reductions. Further increases to $\Delta t = 0.8$ and 1.0 lead to substantial performance degradation. These results indicate robustness to moderate deviations from the step size used during training. In particular, $\Delta t = 0.4$ reduces the maximum step count per attempt to one quarter of the default, with modest score reductions.

6.4 Zero-shot cross-code generalization

Shared local operators allow BeliefFlow to be evaluated on new Tanner graphs without changing the learned weights. Random assignments from separate variable- and check-node feature banks support codes with up to N_{bank} nodes of each type ($N_{\text{bank}} = 128$ by default). Tables 8 and 9 report zero-shot transfer with frozen and learnable banks at $E_b/N_0 = 4\,\text{dB}$, with training codes in columns and test codes in rows. Diagonal entries give the matched-code scores.

The frozen bank transfer results in Table 8 reveal two groups of codes: $\{\text{LDPC}(49, 24)$, $\text{LDPC}(121, 60)$, $\text{LDPC}(121, 70)$, $\text{LDPC}(121, 80)\}$ and $\{\text{MacKay}(96, 48)$, $\text{CCSDS}(128, 64)\}$. Transfer is stronger within each group than across groups. For the first group, this pattern is consistent with the shared array-based LDPC construction (Fan, 2001). Zero-shot transfer may also outperform matched-code baselines: the decoder trained on $\text{LDPC}(121, 60)$ achieves scores of 10.01 and 10.32 on $\text{LDPC}(121, 70)$ and $\text{LDPC}(121, 80)$, respectively, exceeding the matched-code scores of 9.40 and 10.03. Table 9 reports zero-shot transfer results for the learnable-bank variant. The overall transfer pattern resembles that of the frozen-bank variant, but transfer performance is slightly weaker. This may reflect greater specialization of the learned node features to the training code.

Table 8: Frozen-bank zero-shot transfer at $E_b/N_0 = 4\,\text{dB}$ in $-\ln(\text{BER})$. Rows/columns denote test/training codes. Colors show relative differences from diagonal matched-code scores.

	$\text{LDPC}(49, 24)$	$\text{LDPC}(121, 60)$	$\text{LDPC}(121, 70)$	$\text{LDPC}(121, 80)$	$\text{MacKay}(96, 48)$	$\text{CCSDS}(128, 64)$
$\text{LDPC}(49, 24)$	**8.11**	7.83	7.89	7.54	2.99	2.23
$\text{LDPC}(121, 60)$	7.05	**8.24**	7.15	6.37	2.94	1.98
$\text{LDPC}(121, 70)$	8.25	10.01	**9.40**	8.39	3.15	2.22
$\text{LDPC}(121, 80)$	9.24	10.32	10.23	**10.03**	3.53	2.57
$\text{MacKay}(96, 48)$	2.72	2.19	1.97	2.71	**10.21**	9.83
$\text{CCSDS}(128, 64)$	2.81	2.32	2.04	2.75	8.02	**10.77**

Table 9: Learnable-bank zero-shot transfer at $E_b/N_0 = 4\,\text{dB}$ in $-\ln(\text{BER})$. Rows/columns denote test/training codes. Colors show relative differences from diagonal matched-code scores.

	$\text{LDPC}(49, 24)$	$\text{LDPC}(121, 60)$	$\text{LDPC}(121, 70)$	$\text{LDPC}(121, 80)$	$\text{MacKay}(96, 48)$	$\text{CCSDS}(128, 64)$
$\text{LDPC}(49, 24)$	**8.05**	7.58	7.87	7.62	2.72	0.78
$\text{LDPC}(121, 60)$	6.46	**7.65**	7.28	7.00	2.71	0.71
$\text{LDPC}(121, 70)$	7.76	9.63	**9.42**	9.01	2.95	0.71
$\text{LDPC}(121, 80)$	8.54	9.92	10.30	**10.10**	3.38	0.71
$\text{MacKay}(96, 48)$	2.49	3.01	2.01	1.41	**10.20**	9.82
$\text{CCSDS}(128, 64)$	2.57	3.02	2.12	1.15	7.82	**11.36**

7 Limitations

Repeated vector-field evaluations and retries may increase inference cost relative to fixed-depth Transformer decoders such as ECCT and CrossMPT. Moreover, the current formulation of BeliefFlow is tailored to LDPC codes with sparse Tanner graphs. Experiments on Polar and Turbo codes yielded poor decoding performance with this formulation. This is consistent with reported difficulties in applying conventional BP directly to Tanner graphs constructed from the parity-check matrices of these codes (Cammerer et al., 2018; Poulliat et al., 2008).

8 Conclusion

BeliefFlow combines continuous-time Tanner-edge dynamics with expressive local updates and randomized retries. Experiments on six LDPC codes demonstrate strong decoding performance with $0.34\,\text{M}$ parameters, highlighting the potential of BP-structured neural decoding.

AI USE STATEMENT

In this work, we used generative AI tools for polishing the text and improving the overall presentation. These tools also assisted with interpreting experimental results, refining explanations of observed trends, and providing feedback on experimental design and mathematical formulations. We have not used generative AI tools for dataset generation or preprocessing, decoder implementation, or formal proofs, and qualitative and thematic data analysis are not applicable to this work. Additionally, we used generative AI tools for literature searches, manuscript organization and source-code inspection, and auxiliary verification scripts. We have reviewed all AI-assisted work. Specifically, we revised the suggested text and assessed technical descriptions and interpretations against the implementation, experimental results, and cited literature. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Amir Bennatan, Yoni Choukroun, and Pavel Kisilev. Deep learning for decoding of linear codes—a syndrome-based approach. In *IEEE International Symposium on Information Theory*, pp. 1595–1599, 2018. doi: 10.1109/ISIT.2018.8437530. URL https://arxiv.org/abs/1802.04741.

Andreas Buchberger, Christian Häger, Henry D. Pfister, Laurent Schmalen, and Alexandre Graell i Amat. Pruning and quantizing neural belief propagation decoders. *IEEE Journal on Selected Areas in Communications*, 39(7):1957–1966, 2021. doi: 10.1109/JSAC.2020.3041392. URL https://arxiv.org/abs/2011.13594.

Sebastian Cammerer, Moustafa Ebada, Ahmed Elkelesh, and Stephan ten Brink. Sparse graphs for belief propagation decoding of polar codes. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 1465–1469. IEEE, 2018.

Sebastian Cammerer, Jakob Hoydis, Fayçal Aït Aoudia, and Alexander Keller. Graph neural networks for channel decoding. In *IEEE Globecom Workshops*, pp. 486–491, 2022. doi: 10.1109/GCWkshps56602.2022.10008601. URL https://arxiv.org/abs/2207.14742.

Jinghu Chen and Marc P. C. Fossorier. Density evolution for two improved BP-based decoding algorithms of LDPC codes. *IEEE Communications Letters*, 6(5):208–210, 2002. doi: 10.1109/4234.1001666.

Jinghu Chen, Ajay Dholakia, Evangelos Eleftheriou, Marc P. C. Fossorier, and Xiao-Yu Hu. Reduced-complexity decoding of LDPC codes. *IEEE Transactions on Communications*, 53(8):1288–1299, 2005. doi: 10.1109/TCOMM.2005.852852.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018.

Ricky T. Q. Chen, Brandon Amos, and Maximilian Nickel. Learning neural event functions for ordinary differential equations. *International Conference on Learning Representations*, 2021.

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.

Yoni Choukroun and Lior Wolf. Denoising diffusion error correction codes. In *The Eleventh International Conference on Learning Representations*, 2023.

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

Jincheng Dai, Kailin Tan, Zhongwei Si, Kai Niu, Mingzhe Chen, H. Vincent Poor, and Shuguang Cui. Learning to decode protograph LDPC codes. *IEEE Journal on Selected Areas in Communications*, 39(7):1983–1999, 2021. doi: 10.1109/JSAC.2021.3078488. URL https://arxiv.org/abs/2102.03828.

J.R. Dormand and P.J. Prince. A family of embedded runge-kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, 1980. ISSN 0377-0427. doi: https://doi.org/10.1016/0771-050X(80)90013-3. URL https://www.sciencedirect.com/science/article/pii/0771050X80900133.

John L Fan. Array codes as ldpc codes. In *Constrained Coding and Soft Iterative Decoding*, pp. 195–203. Springer, 2001.

Chris Finlay, Joern-Henrik Jacobsen, Levon Nurbekyan, and Adam Oberman. How to train your neural ODE: the world of Jacobian and kinetic regularization. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 3154–3164. PMLR, 13–18 Jul 2020.

Marc P. C. Fossorier, Miodrag Mihaljević, and Hideki Imai. Reduced complexity iterative decoding of low-density parity-check codes based on belief propagation. *IEEE Transactions on Communications*, 47(5):673–680, 1999. doi: 10.1109/26.768759.

Robert G. Gallager. Low-density parity-check codes. *IRE Transactions on Information Theory*, 8 (1):21–28, 1962. doi: 10.1109/TIT.1962.1057683.

Joachim Hagenauer, Elke Offer, Cyril Méasson, and Matthias Mörz. Decoding and equalization with analog non-linear networks. *European Transactions on Telecommunications*, 10(6):659–680, 1999.

Ernst Hairer, Syvert P Nørsett, and Gerhard Wanner. *Solving ordinary differential equations I: Nonstiff problems*. Springer, 1993.

Alon Helvits and Eliya Nachmani. Score based error correcting code decoder. In *Forty-third International Conference on Machine Learning*, 2026.

Saied Hemati and Amir H. Banihashemi. Dynamics and performance analysis of analog iterative decoding for low-density parity-check (LDPC) codes. *IEEE Transactions on Communications*, 54(1):61–70, 2006. doi: 10.1109/TCOMM.2005.861668.

D. E. Hocevar. A reduced complexity decoder architecture via layered decoding of LDPC codes. In *IEEE Workshop on Signal Processing Systems*, pp. 107–112, 2004. doi: 10.1109/SIPS.2004.1363033.

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

Jonathan Kuck, Shuvam Chakraborty, Hao Tang, Rachel Luo, Jiaming Song, Ashish Sabharwal, and Stefano Ermon. Belief propagation neural networks. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/07217414eb3fbe24d4e5b6cafb91ca18-Abstract.html.

Hee-Youl Kwak, Dae-Young Yun, Yongjune Kim, Sang-Hyo Kim, and Jong-Seon No. Boosting learning for LDPC codes to improve the error-floor performance. In *Advances in Neural Information Processing Systems*, volume 36, pp. 22115–22131. Curran Associates, Inc., 2023.

Francois Leduc-Primeau, Saied Hemati, Shie Mannor, and Warren J. Gross. Dithered belief propagation decoding. *IEEE Transactions on Communications*, 60(8):2042–2047, 2012. doi: 10.1109/TCOMM.2012.050812.110115A.

Hyunjae Lee, Yong-Sung Kil, Min Jang, Sang-Hyo Kim, Ok-Sun Park, and Giyoon Park. Multi-round belief propagation decoding with impulsive perturbation for short ldpc codes. *IEEE Wireless Communications Letters*, 9(9):1491–1494, 2020. doi: 10.1109/LWC.2020.2994770.

Haoyu Lei, Chin Wa Lau, Kaiwen Zhou, Nian Guo, and Farzan Farnia. Syndrome-flow consistency model achieves one-step denoising error correction codes. *CoRR*, abs/2512.01389, 2025. doi: 10.48550/arXiv.2512.01389. URL https://arxiv.org/abs/2512.01389.

H.-A. Loeliger, F. Lustenberger, M. Helfenstein, and F. Tarkoy. Probability propagation and decoding in analog vlsi. *IEEE Transactions on Information Theory*, 47(2):837–843, 2001. doi: 10.1109/18.910594.

Loren Lugosch and Warren J. Gross. Neural offset min-sum decoding. In *IEEE International Symposium on Information Theory*, pp. 1361–1365, 2017. doi: 10.1109/ISIT.2017.8006751. URL https://arxiv.org/abs/1701.05931.

M. Moerz, T. Gabara, R. Yan, and J. Hagenauer. An analog 0.25 /spl mu/m bicmos tailbiting map decoder. In *2000 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No.00CH37056)*, pp. 356–357, 2000.

Mohamad Mostafa, Giuseppe Oliveri, Werner G. Teich, and Jürgen Lindner. A continuous-time recurrent neural network for joint equalization and decoding—analog hardware implementation aspects. In Joao Luis Garcia Rosa (ed.), *Artificial Neural Networks—Models and Applications*. IntechOpen, 2016. doi: 10.5772/63387. URL https://www.intechopen.com/chapters/52340.

Eliya Nachmani and Lior Wolf. Hyper-graph-network decoders for block codes. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Eliya Nachmani and Lior Wolf. Autoregressive belief propagation for decoding block codes. *arXiv preprint arXiv:2103.11780*, 2021.

Eliya Nachmani, Yair Be'ery, and David Burshtein. Learning to decode linear codes using deep learning. In *54th Annual Allerton Conference on Communication, Control, and Computing*, pp. 341–346. IEEE, 2016. doi: 10.1109/ALLERTON.2016.7852251. URL https://arxiv.org/abs/1607.04793.

Eliya Nachmani, Elad Marciano, Loren Lugosch, Warren J. Gross, David Burshtein, and Yair Be'ery. Deep learning methods for improved decoding of linear codes. *IEEE Journal of Selected Topics in Signal Processing*, 12(1):119–131, 2018. doi: 10.1109/JSTSP.2017.2788405. URL https://arxiv.org/abs/1706.07043.

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 *The Thirteenth International Conference on Learning Representations*. OpenReview.net, 2025. URL https://openreview.net/forum?id=gFvRRCnQvX.

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*. OpenReview.net, 2026. URL https://openreview.net/forum?id=Xk8cwnwu2e.

Charly Poulliat, David Declercq, and Thierry Lestable. Efficient decoding of turbo codes with nonbinary belief propagation. *EURASIP Journal on wireless communications and networking*, 2008(1):473613, 2008.

Thomas J. Richardson, M. Amin Shokrollahi, and Rüdiger L. Urbanke. Design of capacity-approaching irregular low-density parity-check codes. *IEEE Transactions on Information Theory*, 47(2):619–637, 2001. doi: 10.1109/18.910578.

Eran Sharon, Simon Litsyn, and Jacob Goldberger. Efficient serial message-passing schedules for LDPC decoding. *IEEE Transactions on Information Theory*, 53(11):4076–4091, 2007. doi: 10.1109/TIT.2007.907507.

R. Michael Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547, 1981. doi: 10.1109/TIT.1981.1056404.

Kou Tian, Chentao Yue, Changyang She, Yonghui Li, and Branka Vucetic. A scalable graph neural network decoder for short block codes. In *IEEE International Conference on Communications*, pp. 1268–1273, 2023. doi: 10.1109/ICC45041.2023.10278711. URL https://arxiv.org/abs/2211.06962.

Nedeljko Varnica, Marc P. C. Fossorier, and Aleksandar Kavcic. Augmented belief propagation decoding of low-density parity check codes. *IEEE Transactions on Communications*, 55(7):1308–1317, 2007. doi: 10.1109/TCOMM.2007.900611.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Andres I. Vila Casado, Miguel Griot, and Richard D. Wesel. LDPC decoders with informed dynamic scheduling. *IEEE Transactions on Communications*, 58(12):3470–3479, 2010. doi: 10.1109/TCOMM.2010.101910.070303.

Tadashi Wadayama and Lantian Wei. Gradient flow decoding. *IEEE Access*, 13:131937–131956, 2025. doi: 10.1109/ACCESS.2025.3592296. URL https://arxiv.org/abs/2408.00293.

Hua Xiao and A.H. Banihashemi. Graph-based message-passing schedules for decoding ldpc codes. *IEEE Transactions on Communications*, 52(12):2098–2105, 2004. doi: 10.1109/TCOMM.2004.838730.

Hua Xiao and Amir H. Banihashemi. Comments on successive relaxation for decoding of ldpc codes. *IEEE Transactions on Communications*, 57(10):2846–2848, 2009. doi: 10.1109/TCOMM.2009.10.080005.

A Appendix

A.1 Experiments on training with different solvers

We evaluate Euler and DOPRI5 as alternatives to RK4 during training. Table 10 reports results with matched training and inference solvers on LDPC$(49, 24)$ and MacKay$(96, 48)$ at $E_b/N_0 \in \{4, 5, 6\}$ dB. Euler achieves scores broadly comparable to those of RK4, with differences of at most 0.31 in $-\ln(\mathrm{BER})$. Some DOPRI5 training runs encountered non-finite values during adjoint integration. No such failures were observed with RK4 or Euler. We repeated DOPRI5 training using different random seeds and, for each code, report results from a run that successfully completed the full 800-epoch schedule. These models nevertheless yield lower scores than the default RK4 models trained for the same number of epochs.

Table 10: Decoding performance ($-\ln(\mathrm{BER})$) with matched training and inference solvers. Cell colors indicate relative score differences from RK4/RK4.

	Euler/Euler			RK4/RK4			DOPRI5/DOPRI5		
Code(n, k)	4	5	6	4	5	6	4	5	6
LDPC$(49, 24)$	7.82	10.88	14.48	8.11	11.19	14.54	7.63	10.58	13.37
MacKay$(96, 48)$	10.07	13.30	16.31	10.21	13.33	16.34	9.74	13.10	16.04

A.2 Preliminary results on longer codes

We evaluate BeliefFlow on four longer LDPC codes: WRAN$(384, 320)$, LDPC$(529, 440)$, LDPC$(648, 540)$, and LDPC$(1056, 880)$. To limit computational cost, each model is trained for 100 epochs, one eighth of the default 800-epoch schedule. Each random feature bank contains exact n entries, where n denotes the codeword length. At inference, we use $N_{\mathrm{retry}} = 128$ retry trajectories and a candidate-count stopping threshold of $K = 10$, both half their default values.

As shown in Table 11, BeliefFlow without retries achieves scores broadly comparable to those of flooding BP with at most 200 iterations at $E_b/N_0 = 4\,\mathrm{dB}$, but yields lower scores at $5\,\mathrm{dB}$, particularly on the two longest codes. Retries improve the reported scores across all four codes at both SNRs. With retries, BeliefFlow achieves higher scores than BP on all four codes at $4\,\mathrm{dB}$. At $5\,\mathrm{dB}$, it outperforms BP on LDPC$(529, 440)$ and LDPC$(648, 540)$, achieves a slightly lower score on WRAN$(384, 320)$, and remains below BP on LDPC$(1056, 880)$.

Table 11: Preliminary decoding results on longer LDPC codes, reported as $-\ln(\mathrm{BER})$ at $E_b/N_0 \in \{4, 5\}$ dB. BeliefFlow is trained for 100 epochs; flooding BP uses at most 200 iterations. "BF (no retry)" denotes BeliefFlow without retry decoding.

	BP (200 its)		BF (no retry)		BeliefFlow	
Code(n, k)	4	5	4	5	4	5
WRAN$(384, 320)$	7.76	12.06	7.57	11.17	8.04	12.00
LDPC$(529, 440)$	8.41	15.88	8.68	15.20	10.08	17.28
LDPC$(648, 540)$	8.85	16.54	9.11	14.38	10.73	17.51
LDPC$(1056, 880)$	11.42	17.33	10.86	14.53	12.17	16.47

A.3 Confidence bounds when no bit errors are observed

For evaluations with no observed bit errors, we report a lower confidence bound on $-\ln(\mathrm{BER})$. Let N_{cw} denote the number of evaluated codewords and n the codeword length. The total number of evaluated coded bits is $N_{\mathrm{bit}} = nN_{\mathrm{cw}}$. Under a bitwise binomial model, the bit-error indicators are independent Bernoulli variables with a common error probability $p = \mathrm{BER}$. The total number of bit errors therefore satisfies

$$X \sim \mathrm{Binomial}(N_{\mathrm{bit}}, p), \tag{19}$$

and the probability of observing no errors is

$$\Pr(X = 0 \mid p) = (1 - p)^{N_{\text{bit}}}. \tag{20}$$

For $X = 0$, the exact one-sided $100(1 - \alpha)\%$ upper confidence limit on p is obtained by solving

$$(1 - p_{\text{U}})^{N_{\text{bit}}} = \alpha. \tag{21}$$

Thus,

$$p_{\text{U}} = 1 - \alpha^{1/N_{\text{bit}}}. \tag{22}$$

Since $-\ln(p)$ is monotonically decreasing, this upper limit yields a lower confidence limit on $-\ln(\text{BER})$. Setting $\alpha = 0.05$ gives

$$L_{0.95} = -\ln(p_{\text{U}}) = -\ln\left(1 - 0.05^{1/N_{\text{bit}}}\right). \tag{23}$$

For $N_{\text{cw}} = 1.0 \times 10^8$, the expression yields $L_{0.95} \approx 22.12$ and 22.18 for $n = 121$ and 128, respectively, as reported in Table 1.

It is worth noting that this bound is exact under the bitwise binomial model, which assumes independent bit errors.

A.4 Implementation details of the neural update

This section specifies the normalization, feature transformations, and numerical safeguards omitted from Figure 1.

Feature dimensions and multi-head operations. The message dimension is $d_m = 128$, with $h = 8$ attention heads and $d_h = d_m/h = 16$ features per head. The variable- and check-node stages use separate query, key, and value projections. Head outputs are concatenated along the feature dimension. After check-node aggregation, a linear output projection produces the neural update target f_θ. The linear projections and feed-forward layers use no additive biases.

Normalization and residual feed-forward network. RMS normalization is applied independently to each edge feature:

$$\text{RMSNorm}_j(\boldsymbol{x}) = \boldsymbol{\gamma}_j \odot \frac{\boldsymbol{x}}{\sqrt{d_m^{-1}\|\boldsymbol{x}\|_2^2 + \epsilon_{\text{norm}}}}, \qquad \epsilon_{\text{norm}} = 10^{-5}. \tag{24}$$

Each normalization layer has its own learnable scale $\boldsymbol{\gamma}_j$, initialized to an all-ones vector.

The integrated edge states are normalized before variable-node attention. Let $\boldsymbol{z}_{v\to c}$ denote the concatenated attention output after adding the channel embedding and the projected endpoint features. The intermediate variable-to-check feature is

$$\begin{aligned} \boldsymbol{m}_{v\to c} &= \text{RMSNorm}_2\left(\boldsymbol{z}_{v\to c} + \text{sigmoid}(a)\ \text{FFN}\big(\text{RMSNorm}_1(\boldsymbol{z}_{v\to c})\big)\right), \\ \text{FFN}(\boldsymbol{x}) &= \boldsymbol{W}_2\,\text{GELU}(\boldsymbol{W}_1\boldsymbol{x}). \end{aligned} \tag{25}$$

The FFN has dimensions $128 \to 512 \to 128$, and a is a learnable residual-scaling parameter. No additional FFN is applied within the check-node stage in this configuration.

Attention weights, gates, and offsets. Both stages aggregate over destination-excluded neighborhoods. Variable-node attention uses dot-product scores scaled by $1/\sqrt{d_h}$, followed by a softmax over the aggregation neighborhood. Check-node gates instead use unscaled dot products. For destination edge e, neighboring edge e', and head r,

$$\begin{aligned} g_{e,e'}^{(r)} &= 2\,\text{sigmoid}\left(\boldsymbol{q}_e^{(r)\mathsf{T}}\boldsymbol{k}_{e'}^{(r)}\right), \\ \boldsymbol{u}_{e,e'}^{(r)} &= g_{e,e'}^{(r)}\boldsymbol{v}_{e'}^{(r)}, \\ \boldsymbol{z}_{e,e'}^{(r)} &= \text{sgn}(\boldsymbol{u}_{e,e'}^{(r)}) \odot \max\left(|\boldsymbol{u}_{e,e'}^{(r)}| - \beta, \boldsymbol{0}\right). \end{aligned} \tag{26}$$

The offset is therefore applied after gating. The learnable scalar β is initialized to 0.01 and shared across heads and feature coordinates within the block. No nonnegativity constraint is imposed on β.

Numerical evaluation of the check interaction. Before numerical safeguards, the element-wise check interaction uses the box-plus

$$\boxplus_{i=1}^{s}(z_i) = 2\,\mathrm{atanh}\left(\prod_{i=1}^{s}\tanh(z_i/2)\right), \tag{27}$$

where $\boxplus$ denotes the conventional LLR-domain operator defined in the main text.

Inputs are clipped to $[-10, 10]$ before applying $\tanh$. The resulting values are clipped to $[-1 + \epsilon_{\mathrm{num}}, 1 - \epsilon_{\mathrm{num}}]$, with $\epsilon_{\mathrm{num}} = 10^{-5}$. The product magnitude is evaluated by summing logarithms of absolute values, floored at ϵ_{num}, and then exponentiating. Its sign is determined by the parity of the number of negative inputs. The product is clipped again before applying atanh, and empty aggregation neighborhoods return zero. During inference, the ODE derivative is additionally clipped element-wise to $[-10, 10]$.

A.5 Model size

BeliefFlow uses $0.34\,\mathrm{M}$ parameters at a feature dimension of 128, $N_{\mathrm{bank}} = 128$, compared with approximately $1.2\,\mathrm{M}$ for the six-layer, 128-dimensional ECCT and CrossMPT configurations used for short codes (Choukroun & Wolf, 2022; Park et al., 2025). This corresponds to approximately 72% fewer parameters. Sharing the neural update parameters across integration time allows longer decoding trajectories without increasing the model size.