Structured Masked Diffusion for Shared-Codebook Unsourced Random Access Decoding

Taekyun Lee[1,*] **Jiyoung Yun**[2,*] **Jeffrey G. Andrews**[1] **Hyeji Kim**[1]
[1]Dept. of ECE, The University of Texas at Austin [2]Dept. of ECE, Seoul National University
taekyun@utexas.edu jyyun423@snu.ac.kr
jandrews@ece.utexas.edu hyeji.kim@austin.utexas.edu
[*]Equal contribution.

Abstract

In shared-codebook unsourced random access (URA), a receiver must recover an unordered set of messages from simultaneous transmissions. Its outer decoder assembles ambiguous slot-wise evidence into distinct codewords while enforcing parity constraints. We propose CIDER, a learned outer decoder that combines masked-diffusion refinement with cross-row demixing and parity-aware propagation on the code's Tanner graph. At higher per-bin loads, quality-guided remasking selectively re-decodes low-confidence rows. On the evaluated GF(64) LDPC codes at $K = 2$, CIDER has lower measured symbol and codeword error rates than SIC-BP and FFT-BP at all four code lengths. In same-GPU decoder-only measurements, the deployed mixed-precision implementation is $8.2\times$ to $167\times$ faster than our optimized FFT-BP implementation across those lengths. We also evaluate higher loads, estimated user counts, and detector mismatch. Code is available at https://github.com/jiyunyoung/CIDER/tree/neurips-2026.

1 Introduction

Modern large-scale networked systems—from massive IoT and dense sensing to distributed learning and autonomous systems [Schwarting et al., 2018, Bonawitz et al., 2019, Kalør et al., 2025]—increasingly involve many sporadically active devices with short payloads, making per-device coordination costly. When several devices transmit simultaneously, the receiver must recover multiple coded messages from one noisy aggregate observation [Wang and Poor, 1999, Boutros and Caire, 2002, Polyanskiy, 2017].

Unsourced random access (URA) makes this task particularly challenging: multiple devices share a common codebook and transmit without prior coordination, possibly into the same bin [Polyanskiy, 2017]. The receiver must recover the unordered set of transmitted messages from a noisy mixture without knowing which devices were active or which device sent each codeword. Because the shared-codebook observation carries no per-user labels, decoding requires both assigning plausible slot-wise symbols to distinct message hypotheses and enforcing global code constraints. Practical receivers commonly partition users into bins and perform joint decoding within each bin [Liva and Polyanskiy, 2024, Marshakov et al., 2019, Pradhan et al., 2022]. We focus on this within-bin outer-decoding stage, conditioned on an estimated per-bin load and a known code construction.

The receiver we study has two stages. A symbol-level soft detector maps the raw channel observation to a soft evidence matrix $S \in \mathbb{R}^{L\times Q}$, in which each of the L positions ("slots" in URA) carries a length-Q score vector over candidate symbols. A channel decoder ("decoder" in this paper) then consumes S to recover the unordered set of K transmitted codewords under the code constraints. Because all users share a single codebook and the symbols transmitted in each slot are superposed at the receiver, S aggregates contributions across users and cannot be decoded one user at a time:

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

a *multiuser decoder* is required, which must resolve ownership ambiguity—which (slot, symbol) candidates belong to the same user—while enforcing global codeword consistency under collisions, missed symbols, and false candidates.

Classical multiuser decoders tackle the association problem through three mechanisms. *Joint factor-graph message passing* couples user-separation and code constraints into a single graph and runs BP over all users simultaneously [Pradhan et al., 2022, Amalladinne et al., 2022, Ebert et al., 2022]. *Stitching-based list-recovery* recovers per-slot top candidates and combinatorially assembles valid codewords across slots [Amalladinne et al., 2020, Andreev et al., 2022]. *Successive interference cancellation* (SIC) decodes users in successive stages and subtracts each stage's estimated contribution from the residual evidence [Andreev et al., 2020, Vem et al., 2019, Yun and Choi, 2024]. These methods face distinct limitations. SIC is order-dependent and can propagate early errors. Joint BP may require many iterations with costly non-binary check updates, while the candidate space for stitching-based list recovery grows combinatorially with per-slot ambiguity. In short-packet random access, where decoding latency is a practical constraint, these computational bottlenecks directly limit the practicality of the receiver.

Learned decoders offer another route, but a generic one-shot model does not explicitly encode the permutation invariance or global code constraints of shared-codebook decoding [Choukroun and Wolf, 2024]. Without mechanisms for assigning evidence to distinct hypotheses, it can produce duplicate rows or invalid codewords. A multiuser neural decoder must therefore separate competing user hypotheses while checking that each recovered row is consistent with the code.

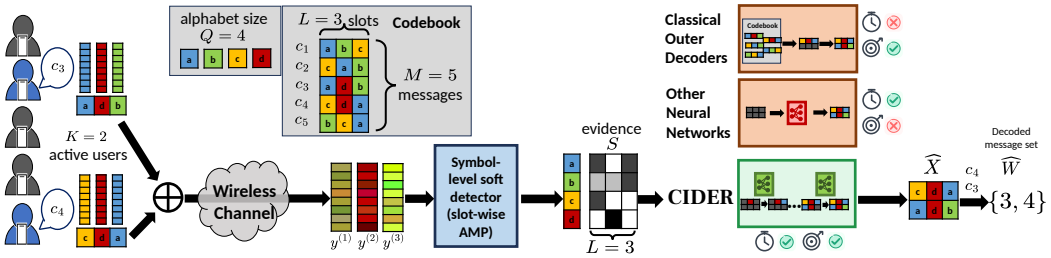


Figure 1: Two-stage receiver pipeline. Active users transmit codewords from a shared codebook over a shared channel. A fixed AMP-based symbol detector converts slot observations into the evidence matrix S, and the multiuser decoder maps S to an unordered set of decoded codewords.

We introduce CIDER (**C**onstraint-aware **I**terative Masked **D**iffusion Decoding for **E**rror-correcting **R**efinement), a learned shared-codebook multiuser decoder. CIDER replaces rule-based decoding with fixed-step masked-diffusion refinement. It turns the ambiguous evidence matrix S into K codewords using two structured operations: demixing, which makes hypothesis rows compete for high-evidence symbols, and parity-aware propagation, which injects sparse code constraints during refinement. This yields a parallel neural analogue of iterative multiuser decoding. We evaluate its reliability and latency against specified neural and classical baselines under a shared evidence interface. Our contributions are as follows.

Contributions.

- We develop a masked-diffusion outer decoder for within-bin, shared-codebook URA. Row competition and parity-aware propagation address duplicate hypotheses and invalid codewords under this set-valued decoding objective.
- We compare with task-aligned classical decoders, tuned neural baselines, and non-diffusion controls on the same evidence interface. The reported error and latency advantages are specific to these implementations and operating points; same-GPU measurements are included in Appendix Section M.
- We provide extensive scaling experiments and modulewise ablations that isolate why the design is necessary (demixing, parity-aware propagation and remasking) and how performance changes with problem size and load.

2 Related Work

Shared-codebook access. Unsourced random access studies recovering an unordered set of messages from a noisy superposition, and is widely analyzed under the Gaussian many-access channel and related models [Polyanskiy, 2017, Kowshik and Polyanskiy, 2021]. Many practical constructions adopt a modular two-stage pipeline, where a front end produces per-slot soft evidence and a later stage assembles code-consistent messages [Amalladinne et al., 2020, Fengler et al., 2021].

Rule-based multiuser decoding. Rule-based multiuser decoders address the set-assembly problem using stitching/list recovery across slots or cancellation/peeling-style procedures combined with iterative decoding [Amalladinne et al., 2020, Andreev et al., 2022, Yun and Choi, 2024, Vem et al., 2019, Ordentlich and Polyanskiy, 2017]. These methods can be effective but may become brittle or expensive under high ambiguity, motivating alternatives that are iterative yet parallelizable.

Learning-based decoding and discrete diffusion. Learned decoders have been explored for channel decoding by augmenting message passing (e.g., neural belief propagation) [Nachmani et al., 2016, 2018] and with attention-based architectures [Choukroun and Wolf, 2022], as well as end-to-end learned coding [Jiang et al., 2019]. In parallel, discrete diffusion and masked denoising models provide iterative refinement mechanisms for categorical variables [Austin et al., 2021, Chang et al., 2022, Lou et al., 2024, Sahoo et al., 2024]. Our work connects these threads by using masked denoising for shared-codebook URA outer decoding. Tarizzo et al. [2025] jointly learn a URA codebook and AMP-style receiver for digital over-the-air aggregation in federated learning. Their receiver targets fragment-level aggregate recovery, whereas our outer decoder recovers complete, distinct codewords from a fixed slot-wise evidence interface. The different receiver outputs and code constructions preclude a like-for-like error-rate comparison. Additional background on wireless random access and related work details are deferred to Appendix Sections B and C.

3 Preliminaries

This section introduces the receiver interface used throughout the paper. We first define the two-stage joint decoding pipeline and the evidence matrix S produced by the fixed symbol-wise soft detector (Section 3.1). We then review masked discrete diffusion for discrete infilling (Section 3.2), which we later use as the basis for our learned multiuser decoder.

Notation. Uppercase symbols denote random variables (or random objects), and the corresponding lowercase symbols denote realizations; we also use calligraphic letters (e.g., $\mathcal{C}, \mathcal{W}$) for sets. Subscripts index components (e.g., v_a, $M_{i,j}$), and parenthesized superscripts denote slot/time indices (e.g., $Y^{(\ell)}$). When a quantity is naturally an entry of a matrix or tensor with a slot dimension, the slot index is written as a subscript alongside the other component indices (e.g., codeword slot symbol $c_{m,\ell}$, evidence entry $S_{\ell,a}$).

3.1 Shared-codebook joint decoding and the two-stage receiver

Figure 1 illustrates the complete two-stage receiver pipeline through a toy example. Suppose two active users transmit the shared-codebook codewords $c_3 = \texttt{adb}$ and $c_4 = \texttt{cda}$, where the alphabet is $\{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}\}$ and the codeword length is $L = 3$. Because the users transmit simultaneously, the receiver does not observe two labeled codewords. Instead, it observes a noisy superposition over the three slots. The **symbol-level soft detector** processes this superposed observation and outputs an evidence matrix $S \in \mathbb{R}^{L \times Q}$, where $S_{\ell,a}$ scores how plausible it is that symbol a appeared in slot ℓ.

The **multiuser decoder** then solves the global assembly problem. Starting from an all-[Mask] grid $X^{(0)} \in ([Q] \cup \{\text{[Mask]}\})^{K \times L}$, CIDER iteratively refines $X^{(t)}$ over T masked-diffusion steps. During this unmasking process, CIDER must decide both which symbols to place and how to assign them to the K user-hypothesis rows. For the example in Figure 1, one valid final decoded grid is $\hat{X} = \begin{bmatrix} \texttt{c} & \texttt{d} & \texttt{a} \\ \texttt{a} & \texttt{d} & \texttt{b} \end{bmatrix}$, up to row permutation. Mapping these rows back to the shared codebook gives the decoded message set $\hat{\mathcal{W}} = \{3, 4\}$. Thus, the detector provides local slot evidence, while the multiuser decoder assembles it into complete, code-consistent messages.

Frame, shared codebook, and unordered target. We study a wireless uplink where many devices send short reports to a common receiver. A frame is divided into L slots, and each active device transmits one symbol per slot, so each message corresponds to a length-L codeword. All devices use the same public codebook $\mathcal{C} = \{c_m\}_{m\in[M]}, c_m \in [Q]^L$, where m is a payload index and $[Q]$ is the Q-ary alphabet. In a given frame, an unknown set of active payloads $\mathcal{W}^\star \subseteq [M]$ with $|\mathcal{W}^\star| = K$ is transmitted simultaneously over the shared channel. The receiver observes a noisy superposition Y of these transmissions and must recover the unordered set of transmitted messages, not their device identities.

Symbol-level soft detector: local slot evidence. The first stage handles the physical-layer mixture. It maps the raw channel observation Y to a slot-wise evidence table $S \in \mathbb{R}^{L\times Q}$, where $S_{\ell,a}$ is the log-posterior score that symbol $a \in [Q]$ is active in slot ℓ. Multiple symbols can be active in one slot. This evidence is local: it tells us which symbols look plausible at each slot, but not which symbols should be grouped together into the same user's codeword. In our experiments, this detector is fixed and implemented by AMP; details are provided in Appendix Section E.

Multiuser decoder: global assembly. The later stage takes only S and produces an unordered decoded grid $\hat{X} \in [Q]^{K\times L}$, whose rows are candidate codewords. Since the target is an unordered message set, the row order of $\hat{X}$ is immaterial. The decoded message set is obtained by mapping valid rows of $\hat{X}$ back to the shared codebook, yielding $\hat{\mathcal{W}} \subseteq [M]$. Thus, the multiuser decoder must solve two coupled problems: it must resolve ownership ambiguity across rows and enforce global code constraints across slots. Our goal is to learn this decoder while keeping the detector and evidence interface S fixed. A full formal model is given in Appendix Section D.

3.2 Masked discrete diffusion for discrete infilling

We briefly review masked (absorbing-state) discrete diffusion, which we use as a conditional discrete infilling mechanism [Chang et al., 2022, Lou et al., 2024, Sahoo et al., 2024]. Starting from a fully masked sequence, the model repeatedly predicts missing tokens and reveals a subset of them over a fixed number of refinement steps. In our joint-decoding setting, the side information is the detector evidence S, and the sequence is reshaped as a $K \times L$ grid.

Masked forward process. Let $[Q] = \{0, 1, \ldots, Q-1\}$ be the token alphabet and let $[\mathsf{Mask}]$ denote an absorbing mask token. For a clean sequence $X_{1:N_{\text{seq}}} \in [Q]^{N_{\text{seq}}}$, the step-$t$ corrupted sequence $\tilde{X}^{(t)} \in ([Q] \cup \{[\mathsf{Mask}]\})^{N_{\text{seq}}}$ is obtained by independently masking each position with probability γ_t, where $\gamma_0 = 1$ and $\gamma_T = 0$:

$$q\Big(\tilde{X}_j^{(t)} \,\Big|\, X_j = x_j\Big) = \begin{cases} 1-\gamma_t, & \tilde{X}_j^{(t)} = x_j, \\ \gamma_t, & \tilde{X}_j^{(t)} = [\mathsf{Mask}], \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Conditional denoising objective. At step t, a denoiser receives the masked sequence $\tilde{x}^{(t)}$, side information ξ, and the step index t, and predicts a categorical distribution $\hat{x}_{\theta,j}(\tilde{x}^{(t)}, \xi, t)$ over $[Q]$ for each position j. Following the standard masked-diffusion objective, we train only on masked positions:

$$\mathcal{L}(\theta) = \mathbb{E}_{t,\tilde{X}^{(t)}\sim q(\cdot|X=x)} \sum_{j\in\Omega_t} \mathrm{CE}\Big(x_j, \hat{x}_{\theta,j}(\tilde{x}^{(t)}, \xi, t)\Big), \tag{2}$$

where $\Omega_t = \{j : \tilde{X}_j^{(t)} = [\mathsf{Mask}]\}$ and $\mathrm{CE}(\cdot,\cdot)$ is categorical cross-entropy [Sahoo et al., 2024].

Inference. At inference, we initialize $\tilde{x}^{(0)} = ([\mathsf{Mask}], \ldots, [\mathsf{Mask}])$ and iteratively reveal high-confidence predictions according to a cosine reveal schedule [Chang et al., 2022]. In CIDER, we set $\xi \equiv S$ and use this refinement process to fill a $K \times L$ decoded grid; implementation details are given in Appendix Section G.2.

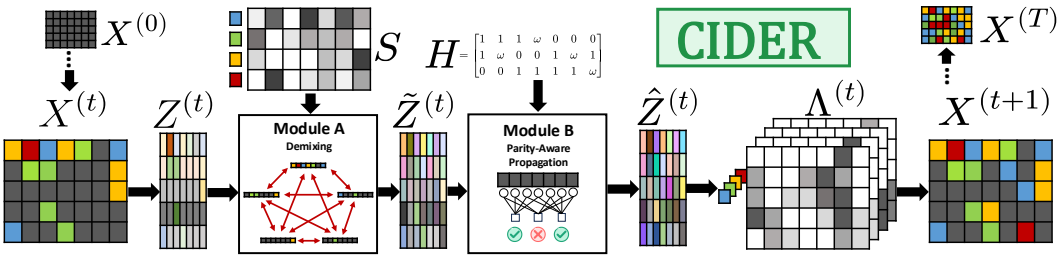


Figure 2: **CIDER structured denoiser.** At refinement step t, the partially revealed grid $X^{(t)}$ is embedded into token latents $Z^{(t)}$. Module A demixes the row hypotheses by making them compete for the shared slot-wise evidence S, producing demixed latents $\tilde{Z}^{(t)}$. Module B then propagates parity information using the parity-check matrix H, producing refined latents $\hat{Z}^{(t)}$ and logits $\Lambda^{(t)}$ over the Q symbols at each grid position. The reveal rule uses $\Lambda^{(t)}$ to update the discrete grid from $X^{(t)}$ to $X^{(t+1)}$, and repeated refinement produces the final decoded grid $X^{(T)} = \hat{X}$.

4 CIDER framework

Design principle. The decoder receives only a slot-wise evidence matrix S: it tells us which symbols are locally plausible in each slot, but not how those symbols should be grouped into K complete codewords. A generic masked-diffusion decoder can refine missing tokens, but it does not know the two structures that make shared-codebook joint decoding difficult: **(A) user hypotheses must be demixed**, so that different rows explain different transmitted codewords rather than collapsing to the same high-evidence symbols, and **(B) each row must be code-consistent**, so that the assembled sequence satisfies the code constraints. CIDER keeps the standard masked-diffusion refinement loop, but replaces the generic per-step denoiser with a structured denoiser designed around these two requirements: **Module A** separates competing rows through evidence-level competition, while **Module B** propagates parity information to guide each row toward a valid codeword.

Overall refinement loop. Figure 2 summarizes the decoder. Starting from an all-[Mask] grid $X^{(0)} \in ([Q] \cup \{[\mathsf{Mask}]\})^{K \times L}$, CIDER repeats a fixed number of refinement steps. At step t, the current grid is embedded into token features, the structured denoiser uses the evidence S and the parity-check matrix H to produce token logits, and the most confident masked entries are revealed:

$$\Lambda^{(t)} = f_\theta(X^{(t)}, S, H, t), \hat{a}^{(t)}_{k,\ell} = \arg\max_{a \in [Q]} \Lambda^{(t)}_{k,\ell,a}, X^{(t+1)}_{k,\ell} = \begin{cases} \hat{a}^{(t)}_{k,\ell}, & (k,\ell) \in \mathcal{U}_t, \\ X^{(t)}_{k,\ell}, & \text{otherwise}, \end{cases}$$

where $\mathcal{U}_t$ is the set of currently masked sites selected for reveal at step t by the confidence-ranked cosine schedule. After T steps, the fully revealed grid $\hat{X} = X^{(T)}$ is mapped back to an unordered decoded message set through the shared codebook. The exact embedding, logit projection, reveal schedule, and first-reveal stabilization rule are given in Appendix Sections G.2 and K.9.

Module A: demixing by row competition. The first failure mode is *duplicate-row collapse*: because the same evidence matrix S is visible to every row, a generic denoiser can assign the same high-evidence slot symbols to multiple user hypotheses. Module A prevents this by turning local evidence into a row-competitive assignment problem. We distinguish between intermediate logits used for demixing and final logits used for token reveal. Before Module A, we compute intermediate logits from the current embedded state,

$$\bar{\Lambda}^{(t)}_{k,\ell,a} = w_a^\top Z^{(t)}_{k,\ell}.$$

These intermediate logits are used only to compute demixing responsibilities. Each slot-symbol candidate (ℓ, a) is softly allocated across the K rows:

$$r^{(t)}_{k,\ell,a} = \frac{\exp(\bar{\Lambda}^{(t)}_{k,\ell,a}/\tau_{\text{demix}})}{\sum_{k'=0}^{K-1} \exp(\bar{\Lambda}^{(t)}_{k',\ell,a}/\tau_{\text{demix}})}, \qquad \sum_{k=0}^{K-1} r^{(t)}_{k,\ell,a} = 1. \tag{3}$$

The responsibility $r^{(t)}_{k,\ell,a}$ measures how much row k claims symbol a in slot ℓ. We then form a row-specific evidence embedding

$$e^{(t)}_{k,\ell} = \sum_{a \in [Q]} r^{(t)}_{k,\ell,a} \, S_{\ell,a} \, v_a, \tag{4}$$

where $v_a \in \mathbb{R}^D$ is a learnable symbol embedding. Finally, this evidence is fused into the token latent,

$$\tilde{Z}_{k,\ell}^{(t)} = \Phi_A\left(Z_{k,\ell}^{(t)}, e_{k,\ell}^{(t)}\right). \tag{5}$$

Thus, high-evidence symbols are not independently copied into every row; they are softly divided among competing rows. This creates a repulsive effect between user hypotheses and encourages different rows to explain different parts of the shared evidence. Implementation details are provided in Appendix Section K.9, with overlap diagnostics in Appendix Section L.10.

Module B: parity-aware propagation for code consistency. The second failure mode is *invalid assembly*: even after rows are separated, each row is still assembled from local slot-wise choices and may violate the code constraints. Module B injects global code structure through the sparse parity-check matrix H. Let $\mathcal{N}_{\mathrm{var}}(j)$ be the slots participating in parity check j, and let $\mathcal{N}_{\mathrm{chk}}(\ell)$ be the checks involving slot ℓ. For each row k, Module B computes an extrinsic check-to-slot signal

$$n_{k,j\to\ell}^{(t)} = \Psi_j\left(\left\{T_{H_{j,\ell'}}\left(\tilde{Z}_{k,\ell'}^{(t)}\right) : \ell' \in \mathcal{N}_{\mathrm{var}}(j) \setminus \{\ell\}\right\}\right), \tag{6}$$

where $T_{H_{j,\ell'}}$ applies the finite-field coefficient action associated with the nonzero parity coefficient $H_{j,\ell'}$, and Ψ_j aggregates the neighboring slot features for check j. In the implementation, Ψ_j uses neighbor-restricted multi-head attention that excludes the target edge, followed by a residual GELU MLP, dropout, and LayerNorm. The Tanner topology and finite-field coefficient permutations are fixed; the message transformations and fusion are learned. The incoming check messages are inverse-permuted, scatter-added, degree-normalized, and fused with the token latent by a learned gate and residual feed-forward network:

$$\hat{Z}_{k,\ell}^{(t)} = \Phi_B\left(\tilde{Z}_{k,\ell}^{(t)}, \sum_{j\in\mathcal{N}_{\mathrm{chk}}(\ell)} n_{k,j\to\ell}^{(t)}\right). \tag{7}$$

After Module B, the final prediction logits are computed from the refined latent state:

$$\Lambda_{k,\ell,a}^{(t)} = w_a^\top \hat{Z}_{k,\ell}^{(t)}.$$

Only these final logits $\Lambda^{(t)}$ are used by the reveal rule during masked-diffusion inference. Each of the N_{layer} structured layers applies Module A and two Tanner message-passing passes within one refinement step; the masked-refinement step count T controls a separate depth. This update is soft: Module B does not hard-project a row onto the codebook. Instead, it biases each refinement step toward completions that are compatible with the parity checks. The finite-field coefficient actions, Tanner-graph aggregation block, and LDPC-code conventions are detailed in Appendix Sections F and K.9; complexity implications are discussed in Appendix Section I.

Why both modules are needed. The two modules solve different parts of the ambiguity. Module A makes the K rows explain different users; without it, rows can duplicate each other even if parity information is present. Module B makes each row obey the code constraints; without it, rows may be distinct but still not valid codewords. The full denoiser therefore combines demixing and code propagation at every refinement step, allowing the masked-diffusion loop to gradually reveal a set of distinct and globally consistent codewords. LayerNorm rescales features but does not assign row ownership; Module A's cross-row softmax couples the hypotheses directly.

Quality-guided remasking at higher loads. For larger K, some decoded rows may remain low-confidence after the first pass. A lightweight quality head is trained post hoc to predict token correctness from the frozen backbone. Its token scores are averaged within each row; low-confidence rows are then remasked and re-decoded while high-confidence rows remain fixed. This PRISM-style [Kim et al., 2026] step is used only as an inference-time reliability boost in higher-load experiments; details are in Appendix Section G.3.

5 Experiments

We evaluate CIDER as a shared-codebook multiuser decoder under a common slot-wise evidence interface. The main experiments compare decoding reliability and latency across LDPC settings, with additional tests of higher per-bin load and a stochastic-binning wrapper. Full setup and implementation details are in Appendix Section H.

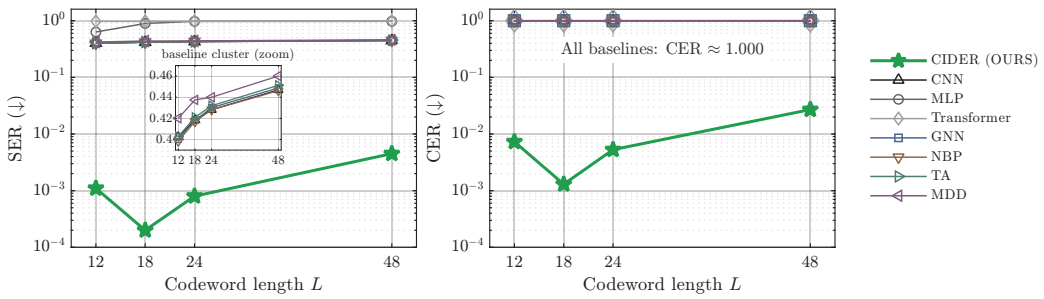


Figure 3: **CIDER vs. neural baselines across code lengths ($K = 2$). Left:** SER vs. code length L at $K = 2$; CIDER achieves low SER, whereas the other neural decoders largely fail and collapse to nearly overlapping total-failure curves. **Right:** CER vs. code length L at $K = 2$. Numerical values are reported in Appendix Section L.1.

5.1 Experimental setup

Unless stated otherwise, experiments use an AWGN channel without per-user fading; a fixed AMP-based detector produces evidence $S \in \mathbb{R}^{L\times Q}$, and all methods decode $S \mapsto \hat{X} \in [Q]^{K\times L}$. The main benchmark uses non-binary LDPC codes over GF(64) with $K = 2$, rate $R = 1/3$, and $L \in \{12, 18, 24, 48\}$. This setting enables fair comparison with strong classical decoders (SIC-BP, FFT-BP, and Top-J search), whose cost already becomes large as K or L grows. We complement it with single-bin scaling up to $K = 8$, stochastic-binning scaling up to $K_{\text{tot}} = 100$, and additional PEG-LDPC/tree-code results. We focus on sparse-graph codes because Module B performs Tanner-graph propagation. Full simulation details are in Appendix Sections H, J, L.9 and L.12.

5.2 Baselines, training, and metrics

We compare CIDER against three groups of decoders under the same evidence interface S: (i) classical decoders (Top-J exhaustive search, SIC-BP, FFT-BP), (ii) one-shot neural decoders (MLP, CNN, Transformer, GNN, NBP, Tanner-Attention), and (iii) generic masked diffusion (MDD). All learning-based models are trained with AdamW on the same train/validation/test splits. At inference, one-shot baselines decode in a single forward pass, while MDD and CIDER run T refinement steps. Full baseline definitions, permutation-invariant training, hyperparameters, and training schedules are provided in Appendix Sections H.6 and K.

Because the decoded message set is unordered, predicted and ground-truth grids are matched by a minimum-Hamming-distance Hungarian assignment before evaluation. Let $\pi^\star$ denote this optimal row permutation. We report symbol error rate (SER) and codeword error rate (CER). For one test example, we compute

$$\mathrm{SER}(X^\star, \hat{X}) = \frac{1}{KL}\sum_{k,\ell} \mathbf{1}\left[\hat{X}_{k,\ell} \neq X^\star_{\pi^\star(k),\ell}\right], \mathrm{CER}(X^\star, \hat{X}) = \frac{1}{K}\sum_{k} \mathbf{1}\left[\hat{X}_{k,:} \neq X^\star_{\pi^\star(k),:}\right]. \quad (8)$$

Reported SER/CER are empirical averages over the test set. Thus, SER evaluates token-level accuracy after row matching, whereas CER evaluates row-level recovery: a decoded codeword is counted as correct only if all L symbols in that row are recovered exactly. The Hungarian matching definition and metric implementation details are given in Appendix Section K.10.

5.3 Main results

Neural baseline comparison. Figure 3 compares CIDER with one-shot neural decoders and generic masked diffusion (MDD) at $K = 2$. Across the four LDPC code lengths, the generic baselines show nearly overlapping high-error curves, while CIDER remains in the low-error regime. The difference is consistent with the need to resolve row ownership and enforce parity when refining a set of codewords from slot-local evidence. Full numerical results are in Appendix Section L.

Classical decoder comparison. Table 1 compares CIDER with SIC-BP, FFT-BP, and Top-J search under the same AMP evidence interface. The table combines SER/CER with optimized same-GPU

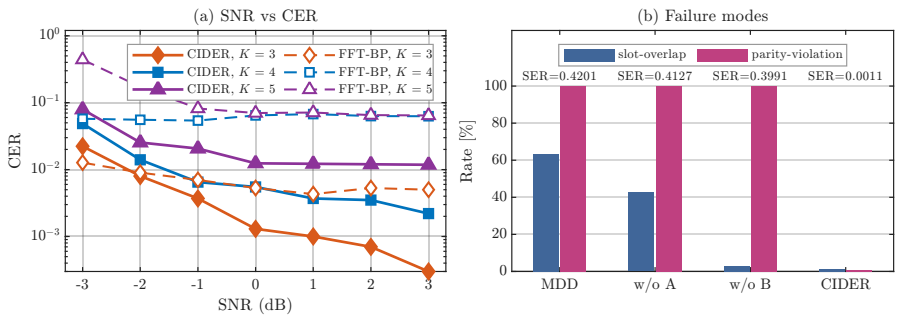


Figure 4: **Classical scaling and CIDER mechanism. Left:** CER (= PUPE) versus per-user per-channel-use SNR. CIDER is competitive at $K = 3$ and shows a growing advantage at higher per-bin loads ($K = 4, 5$); the model is trained at a single operating point (SNR $= -0.79$ dB ($E_b/N_0 = 10$ dB)) and evaluated across the sweep without retraining. Runtime is in Appendix Section L.8. **Right:** Failure-mode diagnostics on Tiny $(Q, L) = (64, 12)$, $K = 2$. Module A primarily resolves duplicate-row overlap; Module B restores code consistency; only the full CIDER achieves both.

latency measurements. CIDER has the lowest SER and CER at all four code lengths and the shortest measured latency among the compared methods. Top-J search becomes costly at larger lengths.

Table 1: Classical decoder accuracy and same-GPU latency under shared AMP evidence ($K = 2$, $Q = 64$). Time is ms/sample on an RTX 3090: BP and Top-J run in FP32 (BP with syndrome early exit) and deployed CIDER uses 16-bit mixed precision. Top-J uses two candidates per slot; at $L = 48$ it exceeded the 60-second cap.

Method	$L = 12$			$L = 18$			$L = 24$			$L = 48$		
	SER	CER	Time	SER	CER	Time	SER	CER	Time	SER	CER	Time
CIDER	**0.0011**	**0.0073**	**1.51**	**0.0002**	**0.0013**	**3.12**	**0.0008**	**0.0053**	**4.31**	**0.0045**	**0.0270**	**10.08**
SIC-BP	0.0015	0.0078	7.29	0.0025	0.0081	14.95	0.0031	0.0076	89.81	0.1144	0.2680	914.15
FFT-BP	0.0015	0.0078	12.38	0.0025	0.0081	27.18	0.0031	0.0076	162.83	0.1144	0.2680	1687.95
Top-J (Top 2)	0.0476	0.0504	2.52	0.0711	0.0760	4.83	0.0911	0.0975	77.61	–	–	$> 60{,}000$

Module ablation. The ablation in Table 2a separates the contributions of masked refinement and the two structural modules. Removing either Module A or Module B drives CER close to one, while generic MDD and non-diffusion controls using the same modules also fall short of full CIDER. Figure 4(b) diagnoses the corresponding failure modes: Module A reduces duplicate-row overlap, and Module B reduces parity violations. Figure 4(a) additionally reports CER across SNRs and per-bin loads.

Runtime. For the same-GPU comparison in Table 1, we vectorized and batched the BP baselines with synchronized FP32 timing and syndrome early exit, and measured deployed 16-bit mixed-precision CIDER on the same RTX 3090 (Appendix Section M.3). At $L = 12/48$, CIDER takes 1.51/10.08 ms/sample, versus 7.29/914.15 for SIC-BP and 12.38/1687.95 for FFT-BP. At the CER $\leq .05$ operating point, $K = 4/7$ CIDER takes 10.19/31.90 ms/sample versus 90.9/525 for SIC-BP; at $K = 8$, SIC-BP does not reach that accuracy target in the tested setting (Appendix Table 30). These are implementation-specific wall-clock results. Batch throughput and memory remain in Appendix Section M.3.

Generalization and scaling. We next evaluate CIDER beyond the $K = 2$ LDPC benchmark. Table 2b reports single-bin results for per-bin loads up to $K = 8$; the low-load error rates are not monotonic across the separately trained K-specific models. At higher loads, PRISM-style quality-guided remasking improves reliability by selectively re-decoding low-confidence rows while clamping high-confidence rows. At $K = 8$, it reduces CER from .2576 to .0403 while increasing end-to-end latency from 30.77 to 63.00 ms/sample, including quality scoring and triggered decoding passes (Appendix Table 33).

On the tree-code instance, CIDER achieves substantially lower SER/CER than the tree-code stitching decoder and Top-J search, while retaining millisecond-scale runtime (Table 2c). This suggests that the gains are not merely due to a particular code construction: even when compared against a code-specific stitching baseline, structured masked refinement provides more reliable accuracy.

We then wrap CIDER in a stochastic-binning protocol [Marshakov et al., 2019, Vem et al., 2019]: each active user randomly selects one of ζ preambles, which assigns its payload to a bin. Following Appendix Section L.12, let K_χ be the load of bin $\chi \in [\zeta]$. We scale ζ with K_{tot} so that $\mathbb{E}[K_\chi] \approx 4$. For this protocol-level evaluation, the receiver is given K_χ and uses the corresponding K-specific CIDER decoder (+PRISM for $K_\chi \geq 6$); bins with $K_\chi > 8$ are counted as erasures. We test estimated loads separately in Table 4. Table 2d reports system-level SER/CER (= PUPE) for K_{tot} up to 100. Further code-family, load-scaling, and protocol details are in Appendix Sections L.6, L.9 and L.12.

Table 2: Compact summary of ablations, load scaling, tree-code baseline comparison, and protocol scaling ($\zeta = \lceil K_{\text{tot}}/4 \rceil$, $\mathbb{E}[K_\chi] \approx 4$). Panels (a)–(c) use Tiny $(Q, L) = (64, 12)$; (d) shows selected protocol loads. The full protocol sweep is in Appendix Table 24.

(a) Module ablation.

Model	**Diff.**	**A**	**B**	**SER**	**CER**
CIDER	✓	✓	✓	**.0011**	**.0073**
w/o A	✓	×	✓	.4127	.9993
w/o B	✓	✓	×	.3991	.9993
MDD	✓	×	×	.4201	.9996
One-shot A+B	×	✓	✓	.1013	.3979
Iter. A+B, N_{iter}=1	×	✓	✓	.1039	.4093
Iter. A+B, N_{iter}=12	×	✓	✓	.3863	.9989

(b) Load scaling (+PRISM).

K	CIDER		+PRISM	
	SER	CER	SER	CER
2	.0011	.0073	–	–
3	.0006	.0044	–	–
4	.0015	.0058	–	–
5	.0048	.0141	–	–
6	.0149	.0349	**.0064**	**.0163**
7	.0441	.1006	**.0096**	**.0247**
8	.1339	.2576	**.0166**	**.0403**

(c) Tree-code baselines.

Method	SER	CER	ms
CIDER	**.0001**	**.0007**	**1.08**
SIC-BP	.0017	.0065	6.82
FFT-BP	.0017	.0065	11.56
Stitch	.0474	.0490	1.20
Top-2	.0474	.0490	2.55

(d) Protocol scaling (selected loads).

K_{tot}	SER	CER
10	.0027	.0108
20	.0225	.0322
40	.0423	.0517
60	.0493	.0590
80	.0502	.0598
100	.0525	.0619

Transfer without weight updates. To test whether CIDER depends on retraining for each code, we apply the $L = 12, K = 2$ checkpoint to different sparse graphs and longer codewords, changing only the masked-refinement step count T (Table 3). PEG-LDPC denotes a graph constructed by progressive edge growth. Transfer CER remains at most .0110 through $L = 24$, then rises to .0762 at $L = 96$; dedicated target-trained models perform better on the other tested settings. The $L = 72, 96$ rows each use 256 test examples. The Training LDPC row uses the Tiny LDPC setting of Table 1 on separately generated on-the-fly test frames.

Decoding with estimated per-bin load. In the stochastic-binning setting above, each bin has a variable number of active users K_χ, which the receiver does not observe directly. Writing K for a bin's true load, we replace the known-K assumption in the preceding single-bin results with an external energy-based estimate $\hat{K}$. Bank uses $\hat{K}$ for AMP and checkpoint selection. Single instead runs one $K = 8$ checkpoint for every bin, deduplicates and ranks its candidate rows by summed slot-wise log-evidence, and retains the top $\hat{K}$ rows. Both modes use the same estimator. Table 4 compares them for $K = 2$–8. After matching the reported and transmitted codeword sets, missed-detection rate (MDR) is the fraction of true codewords missed, and false-alarm rate (FAR) is the fraction of reported codewords that are spurious. The $K = 2$ Bank row uses the Tiny LDPC setting of Table 1 on separately generated on-the-fly test frames; the Single columns evaluate the separate $K = 8$ checkpoint on this load-estimation task.

Table 3: One-checkpoint graph and length transfer at $K = 2$ without weight updates. Only the refinement count T changes. Target-trained results use a dedicated model; $L = 72, 96$ use 256 examples each. Lower CER is better.

Target code	L	T	Transfer CER	Target-trained CER
Training LDPC	12	12	.00807	.00807
PEG-LDPC	12	12	.00783	.00230
Tree code	12	12	.00820	.00070
LDPC	18	16	.0078	.0013
LDPC	24	20	.0110	.0053
LDPC	48	28	.0352	.0270
LDPC	72	40	.0410	—
LDPC	96	60	.0762	—

Table 4: Decoding with estimated per-bin load $\hat{K}$ for true load K. The Single oracle column uses true K only for final row pruning. Lower MDR and FAR are better.

True K	$\Pr(\hat{K} = K)$	Bank		Single		Single oracle
		MDR	FAR	MDR	FAR	MDR
2	.997	.0067	.0080	.0003	.0017	.0003
3	.989	.0067	.0056	.0031	.0016	.0007
4	.959	.0095	.0130	.0045	.0025	.0015
5	.939	.0195	.0197	.0076	.0023	.0016
6	.895	.0224	.0254	.0116	.0084	.0043
7	.853	.0388	.0389	.0203	.0179	.0105
8	.842	.0563	.0458	.0524	.0415	.0435
Mean	.925	.0293	.0278	**.0199**	**.0153**	.0133

Across $K = 2$–8, the estimator has 92.5% mean exact-load accuracy. Single lowers mean MDR/FAR from .0293/.0278 (Bank) to .0199/.0153 but always incurs $K = 8$ computation; oracle pruning lowers its MDR to .0133. A one-user underestimate causes misses, whereas an overestimate adds a false alarm (Appendix Table 25).

AMP and SNR mismatch. To assess sensitivity to the fixed AMP-generated evidence interface, we evaluate CIDER on the Tiny scale without retraining under two types of mismatch: fewer AMP iterations and SNR shifts. Table 26 shows little CER change when AMP iterations change from the matched 10 to 5 or 20 at matched SNR, but CER reaches .3087 under the largest negative SNR shift.

6 Conclusion

We introduced CIDER for within-bin shared-codebook URA outer decoding, where the receiver recovers an unordered set of codewords from fixed slot-wise soft evidence. The key is to refine the unordered codeword set jointly: row competition discourages duplicate rows, while Tanner-graph propagation steers each row toward a valid codeword. The ablations show that both mechanisms are needed. In the evaluated short-block settings, CIDER achieves lower CER than the tested neural baselines; at $K = 2$, it also improves CER and measured outer-decoder latency over the optimized classical baselines on the same GPU.

The results extend beyond one fixed code and load. A single checkpoint transfers across sparse-graph constructions and moderately longer codewords by changing only the refinement count, although target-trained models are usually more accurate. Quality-guided remasking improves reliability at higher per-bin loads at additional runtime cost, and estimated-load decoding shows the importance of load inference. Improving load estimation, demixing under heavy codeword overlap, and robustness to evidence mismatch are the main next steps toward an end-to-end receiver.

Acknowledgements and Disclosures

This work was partly supported by ARO Award W911NF2310062, ONR Award N000142412542, NSF CAREER Award 2443857, NSF RINGS Award CNS-2148141, and the 6G@UT center within the Wireless Networking and Communications Group (WNCG) at the University of Texas at Austin. The authors declare no competing interests.

References

Vamsi Kiran Amalladinne, Jean-François Chamberland, and Krishna R. Narayanan. A coded compressed sensing scheme for unsourced multiple access. *IEEE Transactions on Information Theory*, 66(10):6509–6533, 2020.

Vamsi Kiran Amalladinne, Asit Kumar Pradhan, Cynthia Rush, Jean-François Chamberland, and Krishna R. Narayanan. Unsourced random access with coded compressed sensing: Integrating AMP and belief propagation. *IEEE Transactions on Information Theory*, 68(4):2384–2409, 2022.

Kirill Andreev, Evgeny Marshakov, and Alexey Frolov. A polar code based TIN-SIC scheme for the unsourced random access in the quasi-static fading MAC. In *2020 IEEE International Symposium on Information Theory (ISIT)*, pages 3019–3024, 2020. doi: 10.1109/ISIT44484.2020.9174247.

Kirill Andreev, Pavel Rybin, and Alexey Frolov. Coded compressed sensing with list recoverable codes for the unsourced random access. *IEEE Transactions on Communications*, 70(12):7886–7898, 2022.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 17981–17993, 2021.

Mohsen Bayati and Andrea Montanari. The dynamics of message passing on dense graphs, with applications to compressed sensing. *IEEE Transactions on Information Theory*, 57(2):764–785, 2011.

Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design. *Proceedings of Machine Learning and Systems*, 1:374–388, 2019.

J. Boutros and G. Caire. Iterative multiuser joint decoding: unified framework and asymptotic analysis. *IEEE Transactions on Information Theory*, 48(7):1772–1793, 2002. doi: 10.1109/TIT.2002.1013125.

Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T. Freeman. MaskGIT: Masked generative image transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11315–11325, 2022.

Yoni Choukroun and Lior Wolf. Error correction code transformer. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 38695–38705, 2022. doi: 10.52202/068431-2804.

Yoni Choukroun and Lior Wolf. A foundation model for error correction codes. In *The Twelfth International Conference on Learning Representations*, 2024.

David L. Donoho, Arian Maleki, and Andrea Montanari. Message-passing algorithms for compressed sensing. *Proceedings of the National Academy of Sciences*, 106(45):18914–18919, 2009.

Jamison R. Ebert, Vamsi K. Amalladinne, Stefano Rini, Jean-François Chamberland, and Krishna R. Narayanan. Coded demixing for unsourced random access. *IEEE Transactions on Signal Processing*, 70:2972–2984, 2022. doi: 10.1109/TSP.2022.3182224.

Alexander Fengler, Peter Jung, and Giuseppe Caire. SPARCs for unsourced random access. *IEEE Transactions on Information Theory*, 67(10):6894–6915, 2021.

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

Yihan Jiang, Hyeji Kim, Himanshu Asnani, Sreeram Kannan, Sewoong Oh, and Pramod Viswanath. Turbo autoencoder: Deep learning based channel codes for point-to-point communication channels. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

Anders E. Kalør, Giuseppe Durisi, Sinem Coleri, Stefan Parkvall, Wei Yu, Andreas Mueller, and Petar Popovski. Wireless 6G connectivity for massive number of devices and critical services. *Proceedings of the IEEE*, 113 (9):826–848, 2025.

Malong Ke, Zhen Gao, Mingyu Zhou, Dezhi Zheng, Derrick Wing Kwan Ng, and H Vincent Poor. Next-generation URLLC with massive devices: A unified semi-blind detection framework for sourced and unsourced random access. *IEEE Journal on Selected Areas in Communications*, 41(7):2223–2244, 2023.

Jaeyeon Kim, Seunggeun Kim, Taekyun Lee, David Z. Pan, Hyeji Kim, Sham M. Kakade, and Sitan Chen. Fine-tuning masked diffusion for provable self-correction. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, volume 306 of *Proceedings of Machine Learning Research*, pages 57954–57976, 2026.

Suhas S. Kowshik and Yury Polyanskiy. Fundamental limits of many-user MAC with finite payloads and fading. *IEEE Transactions on Information Theory*, 67(9):5853–5884, 2021.

Gianluigi Liva and Yury Polyanskiy. Unsourced multiple access: A coding paradigm for massive random access. *Proceedings of the IEEE*, 112(9):1214–1229, 2024.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pages 32819–32848, 2024.

Evgeny Marshakov, Gleb Balitskiy, Kirill Andreev, and Alexey Frolov. A polar code based unsourced random access for the gaussian MAC. In *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*, pages 1–5, 2019. doi: 10.1109/VTCFall.2019.8891583.

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

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

Or Ordentlich and Yury Polyanskiy. Low complexity schemes for the random access gaussian channel. In *2017 IEEE International Symposium on Information Theory (ISIT)*, pages 2528–2532. IEEE, 2017.

Mert Ozates, Mohammad Kazemi, and Tolga M. Duman. Unsourced random access using ODMA and polar codes. *IEEE Wireless Communications Letters*, 13(4):1044–1047, 2024. doi: 10.1109/LWC.2024.3359270.

Yury Polyanskiy. A perspective on massive random-access. In *Proc. IEEE International Symposium on Information Theory (ISIT)*, pages 2523–2527, Aachen, Germany, 2017.

Asit Kumar Pradhan, Vamsi K. Amalladinne, Krishna R. Narayanan, and Jean-François Chamberland. Polar coding and random spreading for unsourced multiple access. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pages 1–6, 2020. doi: 10.1109/ICC40277.2020.9148687.

Asit Kumar Pradhan, Vamsi K. Amalladinne, Avinash Vem, Krishna R. Narayanan, and Jean-François Chamberland. Sparse IDMA: A joint graph-based coding scheme for unsourced random access. *IEEE Transactions on Communications*, 70(11):7124–7133, 2022.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 130136–130184, 2024. doi: 10.52202/079017-4135.

Wilko Schwarting, Javier Alonso-Mora, and Daniela Rus. Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):187–210, 2018.

Antonio Tarizzo, Mohammad Kazemi, and Deniz Gündüz. Learned digital codes for over-the-air computation in federated edge learning, 2025. URL `https://arxiv.org/abs/2512.19777`.

Avinash Vem, Krishna R. Narayanan, Jean-François Chamberland, and Jun Cheng. A user-independent successive interference cancellation based coding scheme for the unsourced random access gaussian channel. *IEEE Transactions on Communications*, 67(12):8258–8272, 2019.

Xiaodong Wang and H.V. Poor. Iterative (turbo) soft interference cancellation and decoding for coded CDMA. *IEEE Transactions on Communications*, 47(7):1046–1061, 1999. doi: 10.1109/26.774855.

Jiyoung Yun and Wan Choi. Erasure correcting blind detection in unsourced random access for grant-free massive connections. *IEEE Transactions on Wireless Communications*, 23(3):2428–2439, 2024.

A Notation glossary

This appendix summarizes the main notation used for the receiver, decoder, and experiments; see Table 5.

Table 5: Notation glossary.

Symbol	Meaning
Indices and sizes	
$[M]$	Message index set $\{0, 1, \ldots, M-1\}$; $M \triangleq 2^B$.
$[Q]$	Alphabet/signature index set $\{0, 1, \ldots, Q-1\}$.
$[K]$	Row index set $\{0, 1, \ldots, K-1\}$.
$[L]$	Slot index set $\{0, 1, \ldots, L-1\}$.
$[\zeta]$	Preamble-bin index set $\{1, \ldots, \zeta\}$ in the protocol wrapper.
$k \in [K]$	Row (hypothesis/user) index in the $K \times L$ grid.
$\ell \in [L]$	Slot index; also codeword coordinate index.
$a \in [Q]$	Symbol/signature index within a slot.
$\chi \in [\zeta]$	Bin index in the protocol wrapper.
B, R	Payload length in bits per message and code rate, respectively.
K	Number of active users/messages in one bin; K_χ is the bin's true load and $\hat{K}$ its estimate.
K_{tot}	Total number of active users in a frame (protocol wrapper).
$K_{\max}$	Maximum supported per-bin load (decoder bank is trained up to $K_{\max}$).
ζ	Number of preamble/payload bins in the protocol wrapper.
L	Codeword length, or number of payload slots in one bin.
Q	Alphabet size.
D	Latent/embedding dimension.
P	Number of parity-check equations (rows of H).
N_{seq}	Sequence length in the generic masked-diffusion preliminaries; in the URA grid instantiation, $N_{\mathrm{seq}} = KL$.
Codebook, grids, and decoding	
$\mathcal{C} = \{c_m\}_{m \in [M]}$	Shared codebook.
$c_m \in [Q]^L$	Codeword for message m.
$\mathcal{W}^\star \subseteq [M]$	Ground-truth transmitted message set, $\lvert\mathcal{W}^\star\rvert = K$.
$\hat{\mathcal{W}} \subseteq [M]$	Decoded message set.
$X^\star \in [Q]^{K \times L}$	Ground-truth codeword grid (row order immaterial).
$\hat{X} \in [Q]^{K \times L}$	Decoded codeword grid (row order immaterial).
$X^{(t)} \in ([Q] \cup \{[\mathsf{Mask}]\})^{K \times L}$	Discrete masked grid at CIDER refinement step t (the only persistent state across steps).
γ_t	Mask ratio at step t; $\gamma_0 = 1$, $\gamma_T = 0$.
[Mask]	Mask token.
Symbol-wise soft detector evidence	
$Y^{(\ell)} \in \mathbb{C}^{n_s}$	Random received vector in slot ℓ; n_s is its dimension, $A^{(\ell)} \in \mathbb{C}^{n_s \times Q}$ is the slot signature dictionary, and $y^{(\ell)}$ denotes a realization.
$U^{(\ell)} \in \mathbb{C}^Q$	Slot-ℓ sparse activity vector (random variable).
b_q	q-th standard basis vector in $\mathbb{C}^Q$ (used to form the activity vector $U^{(\ell)}$).
$S_{\ell,a}$	Entry of $S \in \mathbb{R}^{L \times Q}$: log-posterior score that symbol a is active in slot ℓ.
$S^{(\chi)} \in \mathbb{R}^{L \times Q}$	Evidence matrix for bin χ (protocol wrapper).
Diffusion / CIDER	
T	Number of refinement steps.
$Z^{(t)} \in \mathbb{R}^{K \times L \times D}$	Embedded latent constructed from the current discrete grid at step t (recomputed from $X^{(t)}$ each step).
$\tilde{Z}^{(t)} \in \mathbb{R}^{K \times L \times D}$	Demixed latent after Module A at step t.
$\hat{Z}^{(t)} \in \mathbb{R}^{K \times L \times D}$	Refined latent after Module B at step t (used to form logits).
$\Lambda^{(t)} \in \mathbb{R}^{K \times L \times Q}$	Logits output by the denoiser at step t.
$E \in \mathbb{R}^{(Q+1) \times D}$	Embedding table (includes [Mask] token).
$W \in \mathbb{R}^{Q \times D}$	Output projection matrix (rows are $w_a^\top$).
Module A, Module B	Demixing / parity propagation modules.

Continued on next page

Table 5: Notation glossary (continued).

Symbol	Meaning
$r_{k,\ell,a}^{(t)}$	Responsibility (soft assignment) used in demixing.
g_ψ	Per-token quality head used for quality-guided remasking.
$\omega_{k,\ell}^{(t)}$	Per-token correctness score predicted by g_ψ.
$\bar{\omega}_k$	Row-level confidence for row k (average of $\{\omega_{k,\ell}^{(T)}\}_{\ell\in[L]}$ over slots).
$\mathcal{K}_{\text{low}} \subseteq [K]$	Set of remasked (low-confidence) rows in quality-guided remasking at stage s.
φ_s	Quality threshold at remasking stage $s \in \{1, \dots, \varrho\}$.
ϱ	Number of remasking stages (thresholds) in the multi-stage strategy.
$X_{\text{rm}}^{(t)} \in ([Q] \cup \{[\mathsf{Mask}]\})^{K\times L}$	Discrete masked grid during the remasking pass (rows in $\mathcal{K}_{\text{low}}$ are remasked; other rows are clamped).
LDPC / Tanner graph	
$\mathbb{F}_Q$	Finite field of size Q.
$H \in \mathbb{F}_Q^{P\times L}$	Parity-check matrix.
$\mathcal{N}_{\text{var}}(j)$	Variable-node neighbors of check j: $\{\ell : H_{j,\ell} \neq 0\}$.
$\mathcal{N}_{\text{chk}}(\ell)$	Check-node neighbors of variable ℓ: $\{j : H_{j,\ell} \neq 0\}$.
$\alpha \in \mathbb{F}_Q^\times$	Nonzero GF(Q) edge coefficient; $\alpha = H_{j,\ell}$ for edge (j, ℓ).
Π_α	Symbol permutation induced by multiplication by $\alpha \in \mathbb{F}_Q^\times$.
τ_{demix}	Temperature for demixing softmax (Module A).
Matching and metrics	
$\mathfrak{S}_K$	Set of permutations of $[K]$ (row matching).
SER	Symbol error rate (after optimal row matching).
CER	Codeword error rate (after optimal row matching).
Complexity notation	
J	Per-slot candidate list size retained from S (top-J).
$\lvert E_H\rvert$	Number of nonzeros in H (Tanner-graph edges).
I_{BP}	Total Tanner-graph iterations in SIC-BP.

B Wireless communication context and motivation: scalable random access

Grant-based random access and contention. In contemporary cellular systems, uplink transmissions are typically organized around grant-based access. A device first performs random access to establish timing and request an uplink grant, and the base station then schedules that grant for payload transmission. This design works well when the number of simultaneously requesting devices is moderate and payloads are large, because the access overhead can be amortized.

Limitations of contention-based access for low-latency communication. A key challenge of contention-based random access is that it incurs non-negligible access latency due to its reliance on multi-step coordination prior to data exchange. For example, the four-step handshaking procedure in 5G NR requires two round-trip signaling exchanges between a device and the BS to obtain an uplink grant, leading to a baseline access delay that is difficult to amortize for short, latency-critical messages. This latency overhead becomes significantly more severe as the number of simultaneously requesting devices increases: contention opportunities are limited, collision probability on the finite preamble pool rises, and access efficiency degrades due to repeated failures and retransmissions. In latency-stringent regimes, the need to resolve *who transmitted what* through contention resolution and subsequent retransmissions can therefore dominate the end-to-end delay, rendering coordination-first access ill-suited for scalable low-latency communication.

Two-step random access with reduced latency. To reduce access latency and signaling overhead, contemporary wireless systems also support a coordination-based random access procedure with fewer round-trip exchanges in certain regimes. In this design, a device transmits a preamble together with a short message on preconfigured access opportunities using a commonly known coding scheme, and the BS responds only if this concatenated transmission is successfully decoded. While this approach reduces baseline access latency compared to four-step random access, it remains effective primarily under light contention. When multiple devices transmit initial access payloads simultaneously, collisions occur and decoding typically fails, except for occasional capture of the strongest transmission, since current standards do not perform joint multiuser decoding. As a result, this reduced-latency coordination-based access improves latency only when the number of simultaneous access attempts is small, and does not scale to latency-stringent scenarios with many concurrently active devices.

Grant-free access with fixed user–access mapping. Beyond contention-based random access, some systems attempt to reduce handshake overhead by preconfiguring communication resources through device-specific assignments or pilot sequences [Ke et al., 2023]. Under these designs, each potential device is associated with a distinct assignment, signature, or codebook ahead of time, so decoding proceeds under a fixed mapping between received observations and users. While effective in relatively static settings, this approach scales poorly with a large population of potential users and unpredictable activity. Maintaining per-user signatures, codebooks, or scheduling state can incur substantial overhead, since preallocated resources are wasted whenever only a fraction of devices are active. Moreover, rapid changes in the device set require continual reconfiguration, rendering fixed user–resource mappings ill-suited for scalable low-latency communication.

Motivation for URA. Latency-stringent systems with many potential users often cannot afford the overhead of explicit handshaking, scheduling, or repeated access attempts prior to data transmission. In such regimes, transmitting payloads without prior coordination provides a scalable operating point, allowing many users to access the channel simultaneously while avoiding per-user access overhead. This operating principle fundamentally changes how random access should be modeled and decoded. The unsourced random access (URA) abstraction captures this regime by formulating random access as a joint coding and inference problem rather than a coordination-first protocol. When payloads are transmitted without prior coordination, per-user allocation and transmitter identification cannot be assumed during decoding. As a result, URA defines the receiver's objective as recovering the set of transmitted messages directly from the superposed uplink signal. By avoiding per-user preallocation and coordination dependencies, URA decouples system performance from the total number of potential users, making it particularly well suited for large, dynamic systems with sporadic activity and strict latency constraints. From a physical-layer perspective, this amounts to using the random access channel itself as a data transmission channel, rather than merely as a means for access coordination.

Decoding implications of uncoordinated access. Operating without prior coordination has two key implications for the decoding problem. First, since the receiver cannot rely on device-specific configuration or scheduling and does not know which devices are active at the time of decoding, all devices must use a commonly known codebook. This shared encoding rule ensures that the joint decoding is well defined even when the active set is unknown. Second, since the receiver lacks prior knowledge of which devices are active before data exchange, decoding is naturally formulated in terms of message-set recovery rather than message-to-transmitter association. The transmitter identity is therefore unknown prior to transmission and, if necessary, may be embedded within the payload and recovered only after successful message decoding.

C Additional related work details

URA problem context. Unsourced random access (URA) is commonly studied as a model for massive sporadic uplink access, including formulations based on the Gaussian many-access channel (GMAC) [Polyanskiy, 2017]. Within this line of work, a range of URA constructions and analyses have been developed, including coded compressed sensing pipelines [Amalladinne et al., 2020], SPARCS-type schemes [Fengler et al., 2021], and information-theoretic characterizations [Kowshik and Polyanskiy, 2021].

C.1 URA architectural families

The recent survey [Liva and Polyanskiy, 2024] classifies URA receivers into four architectural families:

- **Multipacket-reception (MPR) slotted-Aloha schemes.** Each frame is divided into slots, and within each slot a small group of colliding users is jointly decoded rather than treated as destructive collisions. Examples include T-fold polar Aloha [Marshakov et al., 2019] and T-fold LDPC schemes [Vem et al., 2019], often combined with SIC across slots.
- **Coded compressed sensing (CCS).** Messages are split into fragments, each compressed via a per-slot signature dictionary, and recovered fragments are assembled into valid codewords using a tree code or list-recoverable code [Amalladinne et al., 2020, Fengler et al., 2021, Andreev et al., 2022]. Joint AMP and multiuser-code message passing has also been integrated within this family [Amalladinne et al., 2022, Ebert et al., 2022].
- **Preamble-based architectures.** The user message is split into two parts: the first selects a preamble that determines a per-user resource or access pattern, and the second is encoded with a channel code. Decoding is then performed within each bin, using either joint multiuser decoding (e.g., sparse IDMA [Pradhan et al., 2022]) or single-user decoders aided by SIC [Ozates et al., 2024].
- **Spreading-based architectures.** Message bits select spreading sequences from a dictionary, and recovered codewords are extracted by energy detection followed by per-user channel decoding with SIC [Pradhan et al., 2020].

Learning for decoding: connections to channel decoding and URA. Beyond these rule-based multiuser decoders, learning-based decoders have been studied for classical channel decoding by augmenting iterative message passing, including neural belief propagation and learned Tanner-graph updates [Nachmani et al., 2016, 2018], and by using attention-based architectures for soft decoding [Choukroun and Wolf, 2022]. End-to-end learned coding has also been explored, where encoder and decoder are jointly trained (e.g., TurboAE) [Jiang et al., 2019]. These approaches are typically formulated for decoding a single codeword with a fixed observation-to-codeword association. In URA, the decoding target is an *unordered list* (set) of messages (unsourced decoding), and the receiver must resolve ownership ambiguity across users in addition to enforcing code constraints [Polyanskiy, 2017, Amalladinne et al., 2020, Kowshik and Polyanskiy, 2021].

Discrete diffusion and masked denoising models. Discrete diffusion models have been developed for categorical variables. D3PM defines a Markov noising process that gradually corrupts a discrete token by mixing it with a base categorical distribution (often uniform over the vocabulary), and learns a reverse-time model that predicts the corresponding denoising transitions [Austin et al.,

2021]. In contrast, masked denoising models (MDM), also described as masked-token diffusion, use an absorbing mask token and train with a masked prediction objective, enabling reconstruction via iterative unmasking procedures [Chang et al., 2022, Lou et al., 2024, Sahoo et al., 2024, Kim et al., 2026]. Both formulations provide a principled way to perform *iterative refinement* of discrete hypotheses under uncertainty, which matches the role of shared-codebook multiuser decoding: the slot-wise soft detector produces noisy, incomplete per-slot symbol evidence, and the decoding stage refines it into a globally consistent discrete output under code constraints. We build our decoder on the masked denoising (MDM) formulation; additional uniform-corruption (D3PM-style) comparisons are deferred to Appendix Section L.3.

D Formal URA model and two-stage decoding interface

This appendix provides a formal description of the URA two-stage receiver abstraction used throughout the paper. We define the slot-wise codebook interface, the per-slot observation model, the evidence matrix produced by the fixed symbol-level soft detector, and the multiuser decoder objective as set recovery under permutation invariance.

D.1 URA system model (frame-level set recovery)

Messages and unsourced objective. Let B be the payload length in bits and $M \triangleq 2^B$ be the number of possible messages. We index messages by $[M] \triangleq \{0, 1, \ldots, M-1\}$. In one decoding window (frame), an unknown subset of users becomes active and transmits a subset of messages

$$\mathcal{W}^\star \subseteq [M], \qquad |\mathcal{W}^\star| = K,$$

where K is the number of active messages in the frame (often treated as known in evaluation settings). In *unsourced* random access, user identities are not part of the decoding objective; the receiver outputs only the unordered set $\hat{\mathcal{W}}$. In particular, the receiver is not required to determine *which* device sent a decoded message. (If two devices transmit the same payload, the observation is indistinguishable from a single transmission of that payload under the set-valued objective; when M is large this event is typically negligible.)

D.2 Slot grid and shared codebook

Slot decomposition. We represent each message by a length-L sequence over a Q-ary alphabet $[Q] \triangleq \{0, 1, \ldots, Q-1\}$. The shared codebook is

$$\mathcal{C} = \{c_m\}_{m \in [M]}, \qquad c_m \in [Q]^L.$$

The ℓ-th slot symbol of message m is denoted by $c_{m,\ell} \in [Q]$. Equivalently, $c_{m,\ell}$ is an *index* selecting one of Q pre-defined per-slot signatures/waveforms in slot ℓ.

Set-valued codeword representation. Let $(w_0, \ldots, w_{K-1})$ be an arbitrary ordering of $\mathcal{W}^\star$. The corresponding transmitted codeword set is

$$\mathcal{C}^\star \triangleq \{c_m : m \in \mathcal{W}^\star\}.$$

For notational convenience, we represent the same unordered set $\mathcal{C}^\star$ by a grid $X^\star \in [Q]^{K \times L}$ whose rows are the K codewords in any order:

$$X^\star_{k,\ell} \triangleq c_{w_k,\ell}.$$

Any row permutation of $X^\star$ represents the same solution.

D.3 Per-slot channel model and symbol-level soft evidence

Signature dictionary per slot. For each slot $\ell \in [L]$, each symbol $q \in [Q]$ corresponds to a known waveform (signature) $a_q^{(\ell)} \in \mathbb{C}^{n_s}$. Collecting them forms a dictionary

$$A^{(\ell)} \triangleq \left[a_0^{(\ell)} \ \cdots \ a_{Q-1}^{(\ell)}\right] \in \mathbb{C}^{n_s \times Q}.$$

Superposition observation. In slot ℓ, the receiver observes a superposition of the K active signatures plus noise:

$$Y^{(\ell)} \;=\; \sum_{k=0}^{K-1} a^{(\ell)}_{c_{w_k,\ell}} \;+\; \epsilon^{(\ell)}, \qquad \epsilon^{(\ell)} \sim \mathcal{CN}(0, \sigma^2 I). \tag{9}$$

This is a simplified (yet standard) URA abstraction in which each active transmission contributes one unit-amplitude signature per slot. More general models include per-user (or per-slot) complex gains multiplying each selected signature; our experiments use the above normalization for simplicity. Equivalently, define a sparse activity vector $U^{(\ell)} \in \mathbb{C}^Q$ by

$$U^{(\ell)} \triangleq \sum_{k=0}^{K-1} b_{c_{w_k,\ell}}, \tag{10}$$

where b_q is the q-th standard basis vector. Then (9) can be rewritten as

$$Y^{(\ell)} = A^{(\ell)} U^{(\ell)} + \epsilon^{(\ell)}. \tag{11}$$

To see (11) directly, note that $A^{(\ell)} b_{c_{w_k,\ell}} = a^{(\ell)}_{c_{w_k,\ell}}$ by definition of the dictionary columns, hence

$$A^{(\ell)} U^{(\ell)} = A^{(\ell)} \sum_{k=0}^{K-1} b_{c_{w_k,\ell}} = \sum_{k=0}^{K-1} A^{(\ell)} b_{c_{w_k,\ell}} = \sum_{k=0}^{K-1} a^{(\ell)}_{c_{w_k,\ell}}.$$

Collisions correspond to multiple users selecting the same index and summing at the same coordinate of $U^{(\ell)}$. When collisions occur, the corresponding entry $U^{(\ell)}_a$ becomes an integer count (or, under more general fading models, an aggregated complex amplitude).

Evidence interface S. The symbol-level soft detector is fixed throughout this paper. For each slot ℓ, it maps $(Y^{(\ell)}, A^{(\ell)})$ to a real-valued score vector over $[Q]$. We define log-posterior evidence (logits)

$$S_{\ell,a} \triangleq \log \Pr\Big(U^{(\ell)}_a \neq 0 \,\Big|\, Y^{(\ell)}, A^{(\ell)}\Big), \qquad a \in [Q],\ \ell \in [L].$$

Equivalently, normalizing over a yields per-slot soft beliefs

$$p_{\ell,a} \;\propto\; \exp(S_{\ell,a}), \qquad a \in [Q],$$

which we use as a normalized per-slot categorical belief for the downstream decoder. Since multiple symbols can be active in one slot, this softmax normalization should not be interpreted as a calibrated multi-hot activity probability. Stacking all slots yields an evidence matrix $S \in \mathbb{R}^{L \times Q}$. All multiuser decoders compared in this paper operate only on S (the symbol-level soft detector is not modified). Details of the AMP–MMSE detector used in our implementation are given in Appendix Section E.

D.4 Multiuser decoding objective and set-valued output

Multiuser decoder as a mapping from evidence to a $K \times L$ grid. The multiuser decoding stage is modeled as a mapping

$$\hat{X} = \mathcal{G}(S), \qquad \hat{X} \in [Q]^{K \times L},$$

where $\hat{X}$ is defined up to row permutation. Many implementations of $\mathcal{G}$ produce per-site logits $\Lambda \in \mathbb{R}^{K \times L \times Q}$ and then decode by argmax

$$\hat{X}_{k,\ell} = \arg\max_{a \in [Q]} \Lambda_{k,\ell,a}.$$

In our learned decoders (MDD/CIDER), the logits Λ arise from a masked-denoising refinement loop conditioned on S. In rule-based baselines, Λ may be produced implicitly (e.g., via BP marginals) or explicitly as per-slot posterior scores.

Mapping a decoded grid to a message set. Given $\hat{X}$, the decoded message set is

$$\hat{\mathcal{W}} \;\triangleq\; \Big\{ m \in [M] \;:\; \exists\, k \in [K] \text{ s.t. } \hat{X}_{k,:} = c_m \Big\}.$$

That is, we keep the message indices whose codewords appear among the decoded rows. (When $\mathcal{C}$ is an LDPC code, this corresponds to keeping only parity-valid rows under the shared codebook.) In practice, the "row $\mapsto$ message" step can be implemented either by explicit lookup in $\mathcal{C}$ (when the full codebook is enumerated) or by checking code membership/validity (e.g., $H\hat{X}_{k,:} = 0$ for an LDPC code) together with a payload-to-codeword convention.

Permutation invariance and row matching. Because URA is unsourced, row order is meaningless. When comparing $\hat{X}$ and $X^\star$, we first align rows by an optimal permutation (Hungarian matching). Let $\mathfrak{S}_K$ denote the set of permutations of $[K]$. We define

$$\pi^\star \triangleq \arg\min_{\pi\in\mathfrak{S}_K} \sum_{k=0}^{K-1}\sum_{\ell=0}^{L-1} \mathbf{1}\left[\hat{X}_{k,\ell} \neq X^\star_{\pi(k),\ell}\right]. \tag{12}$$

All symbol- and codeword-level metrics in the main text are computed after this alignment. Equivalently, (12) maximizes the total number of matched symbol positions under a one-to-one assignment between predicted and true rows.

What this paper replaces. Classical URA pipelines implement $\mathcal{G}$ using rule-based stitching and sequential SIC-style peeling under code constraints. In this paper, we keep the symbol-level soft detector and evidence interface S unchanged and replace the mapping $\mathcal{G}$ with our masked-diffusion-based decoder (CIDER).

E Slot-wise AMP–MMSE detector

We use a fixed detector that operates independently for each slot $\ell \in [L]$. Within a slot, the receiver faces a sparse inverse problem: among Q candidate signatures, only a small subset actually appears, because each of the K active devices selects (at most) one signature index in that slot. We adopt Approximate Message Passing (AMP) with an MMSE denoiser to produce *soft* activity evidence [Donoho et al., 2009, Bayati and Montanari, 2011].

Slot observation model (why it is sparse recovery). In slot ℓ, we observe a length-n_s received vector $Y^{(\ell)} \in \mathbb{C}^{n_s}$:

$$Y^{(\ell)} = A^{(\ell)}U^{(\ell)} + \epsilon^{(\ell)}, \qquad \epsilon^{(\ell)} \sim \mathcal{CN}(0, \sigma^2 I).$$

Here $A^{(\ell)} \in \mathbb{C}^{n_s\times Q}$ is a known sensing/dictionary matrix whose Q columns correspond to the Q possible signatures in slot ℓ (Partial-DFT in our experiments). The unknown vector $U^{(\ell)} \in \mathbb{C}^Q$ encodes which signatures participated in the slot: the coordinate $U_a^{(\ell)}$ is *active* if signature index $a \in [Q]$ was used by at least one active device in slot ℓ, and *inactive* if that signature was not used in the slot. Equivalently, "active" means $U_a^{(\ell)} \neq 0$ and "inactive" means $U_a^{(\ell)} = 0$. Since typically $K \ll Q$, only a small fraction of the Q coordinates are active, so $U^{(\ell)}$ is sparse. When $n_s < Q$, recovering $U^{(\ell)}$ from $Y^{(\ell)}$ is underdetermined unless we exploit this sparsity. In the simplified URA model in Equation (10), collisions make some active coordinates take integer values larger than 1; AMP remains applicable because it performs approximate Bayesian inference under a continuous-valued sparse prior.

Probabilistic sparsity model. To produce calibrated *soft* evidence (rather than only a hard support estimate), we use a Bernoulli–Gaussian prior on each coordinate:

$$U_a^{(\ell)} \;\sim\; (1-\rho_{\mathrm{BG}})\,\delta_0 \;+\; \rho_{\mathrm{BG}}\,\mathcal{CN}(0,\sigma_u^2), \qquad a \in [Q].$$

Under this model, ρ_{BG} controls the probability that a given signature coordinate is active (i.e., nonzero) in a slot, and σ_u^2 sets the typical magnitude scale of a nonzero (active) coordinate. In a URA slot, ρ_{BG} is conceptually tied to the expected sparsity level: if there are K active devices choosing among Q signatures, then only on the order of K coordinates are expected to be active (with collisions potentially reducing the number of distinct active coordinates). We treat $(\rho_{\mathrm{BG}}, \sigma_u^2)$ as fixed symbol-wise soft detector hyperparameters; the multiuser decoder sees only the resulting evidence matrix S.

Why AMP and what the iterations represent. AMP is an iterative method designed for large sensing matrices where repeated linear updates can be approximated by a "signal + effective Gaussian noise" scalar channel. At a high level, AMP alternates between: (i) forming a per-coordinate pseudo-observation by backprojecting the current residual through $(A^{(\ell)})^H$, and (ii) applying a scalar denoiser coordinate-wise to update the estimate. Crucially, AMP includes an additional correction (the Onsager correction) in the residual update, which cancels the dominant self-interference created

by reusing the same sensing matrix across iterations; this stabilization is what makes the scalar "effective noise" view accurate in practice. When $A^{(\ell)}$ has approximately orthogonal columns (as with partial-DFT) and the problem dimensions are moderate-to-large, this approximation is accurate enough that a simple per-coordinate denoiser yields useful posterior estimates.

AMP–MMSE: denoiser and iterations. AMP is built around a simple scalar view of each coordinate: it treats a *pseudo-observation* as if it were a noisy scalar measurement

$$R \;=\; U + W, \qquad W \sim \mathcal{CN}(0, \nu),$$

and then applies the *scalar MMSE denoiser*

$$\eta_{\mathrm{MMSE}}(r; \nu) \;\triangleq\; \mathbb{E}[U \,|\, R = r]\,. \tag{13}$$

AMP also uses a companion *sensitivity* term $\eta'_{\mathrm{MMSE}}(r; \nu)$ to scale the Onsager correction; in real-valued notation, one may view it as the derivative of the denoiser,

$$\eta'_{\mathrm{MMSE}}(r; \nu) \;\triangleq\; \frac{\partial}{\partial r}\, \eta_{\mathrm{MMSE}}(r; \nu), \tag{14}$$

and in the complex setting we use the standard AMP analogue of this scalar sensitivity (see Bayati and Montanari [2011]).

In our symbol-wise soft detector, the prior on U is Bernoulli–Gaussian: $U \sim (1-\rho_{\mathrm{BG}})\delta_0 + \rho_{\mathrm{BG}}\mathcal{CN}(0, \sigma_u^2)$. Here $U \neq 0$ ("present") means that the corresponding signature index actually participated in the slot, i.e., $U_a^{(\ell)} \neq 0$ for that coordinate a. Under this prior and the scalar model above, the posterior activity probability $\hat{p}(r) = \Pr(U \neq 0 \mid R = r)$ and the MMSE denoiser (13) admit closed forms:

$$\hat{p}(r) = \frac{\rho_{\mathrm{BG}}\, \phi(r; 0, \nu + \sigma_u^2)}{(1-\rho_{\mathrm{BG}})\, \phi(r; 0, \nu) \;+\; \rho_{\mathrm{BG}}\, \phi(r; 0, \nu + \sigma_u^2)}, \tag{15}$$

$$\eta_{\mathrm{MMSE}}(r; \nu) = \hat{p}(r) \cdot \frac{\sigma_u^2}{\nu + \sigma_u^2}\, r, \tag{16}$$

where $\phi(\cdot; 0, \cdot)$ denotes the circular complex Gaussian density. For completeness, the activity posterior in (15) follows from Bayes' rule using the mixture prior:

$$\Pr(U \neq 0 \mid R = r) = \frac{\rho_{\mathrm{BG}}\, p(r \mid U \neq 0)}{(1-\rho_{\mathrm{BG}})\, p(r \mid U = 0) + \rho_{\mathrm{BG}}\, p(r \mid U \neq 0)},$$

with $p(r \mid U = 0) = \phi(r; 0, \nu)$ and $p(r \mid U \neq 0) = \phi(r; 0, \nu + \sigma_u^2)$ after marginalizing $U \sim \mathcal{CN}(0, \sigma_u^2)$.

The MMSE mean (16) then combines the posterior mean under the Gaussian component, $\mathbb{E}[U \mid R = r, U \neq 0] = \frac{\sigma_u^2}{\nu + \sigma_u^2} r$, with the activity probability $\hat{p}(r)$. In practice, AMP evaluates (16) coordinate-wise, and uses the corresponding sensitivity term (14) (or its standard AMP complex counterpart) inside the Onsager correction.

With these definitions, for each slot ℓ AMP maintains an estimate $\hat{U}^{(i)} \in \mathbb{C}^Q$ of $U^{(\ell)}$ and a measurement-domain residual $\tilde{Y}^{(i)} \in \mathbb{C}^{n_s}$ (the part of $Y^{(\ell)}$ not yet explained by the current estimate). We initialize $\hat{U}^{(0)} = 0$ and $\tilde{Y}^{(0)} = Y^{(\ell)}$, and for $i = 0, 1, \ldots, I_{\mathrm{AMP}} - 1$ we iterate:

$$R^{(i)} = \hat{U}^{(i)} + (A^{(\ell)})^H \tilde{Y}^{(i)}, \tag{17}$$

$$\hat{U}_a^{(i+1)} = \eta_{\mathrm{MMSE}}\Big(R_a^{(i)};\, \nu^{(i)}\Big) \qquad (a \in [Q]), \tag{18}$$

$$\tilde{Y}^{(i+1)} = Y^{(\ell)} - A^{(\ell)}\hat{U}^{(i+1)} \;+\; \frac{\tilde{Y}^{(i)}}{n_s} \sum_{a \in [Q]} \eta'_{\mathrm{MMSE}}\Big(R_a^{(i)};\, \nu^{(i)}\Big)\,, \tag{19}$$

where $\nu^{(i)}$ is an effective noise-variance estimate for the pseudo-observation (e.g., $\nu^{(i)} \approx \|\tilde{Y}^{(i)}\|_2^2 / n_s$). The final term in (19) is the Onsager correction, which uses the denoiser sensitivity $\eta'_{\mathrm{MMSE}}(\cdot)$ to compensate for iteration-to-iteration correlations when reusing $A^{(\ell)}$ [Bayati and Montanari, 2011].

Evidence interface. After I_{AMP} iterations, AMP provides a final pseudo-observation $R^{(I_{\mathrm{AMP}})} \in \mathbb{C}^Q$ (and an associated effective variance estimate $\nu^{(I_{\mathrm{AMP}})}$). We then convert these into a slot-wise evidence heatmap by evaluating the posterior activity probability under the Bernoulli–Gaussian scalar model:

$$S_{\ell,a} \triangleq \log \Pr\big(U_a^{(\ell)} \neq 0 \mid Y^{(\ell)}, A^{(\ell)}\big) \approx \log \hat{p}\Big(R_a^{(I_{\mathrm{AMP}})}; \nu^{(I_{\mathrm{AMP}})}\Big),$$

where $\hat{p}(\cdot;\cdot)$ is given in (15). Stacking over slots yields $S \in \mathbb{R}^{L\times Q}$. Throughout the paper, the symbol-wise soft detector (and hence S) is fixed, and we train/compare the multiuser decoder.

F Q-ary LDPC code

Our main experiments use a *Q-ary* low-density parity-check (LDPC) code over the finite field $\mathbb{F}_Q$ (here $Q = 64$) to impose *global consistency across slots*. In the main text, we denote the shared URA codebook by $\mathcal{C} = \{c_m\}_{m\in[M]}$ with codewords $c_m \in [Q]^L$. In our LDPC setting, we fix a bijection $\phi : [Q] \to \mathbb{F}_Q$ and use it to interpret each discrete symbol index as a field element. With a slight abuse of notation, we write $c_m \in \mathbb{F}_Q^L$ (equivalently $\mathbf{c}_m = \phi(c_m) \in \mathbb{F}_Q^L$) when discussing parity constraints. In this regime, the message index set $[M]$ can be viewed as an enumeration of the LDPC codebook; when H has full row rank, $|\mathcal{C}_{\mathrm{LDPC}}| = Q^{L-P}$ and thus $B = \log_2 |\mathcal{C}_{\mathrm{LDPC}}|$ (so $M = 2^B$).

Linear code definition via a sparse parity-check matrix. A Q-ary LDPC code is a linear block code specified by a sparse parity-check matrix $H \in \mathbb{F}_Q^{P\times L}$. The set of valid codewords is

$$\mathcal{C}_{\mathrm{LDPC}} \triangleq \{\, \mathbf{c} \in \mathbb{F}_Q^L : H\mathbf{c} = \mathbf{0} \text{ in } \mathbb{F}_Q \,\}. \tag{20}$$

In our experiments, the codebook $\mathcal{C}$ is chosen to be (or to be identified with) this LDPC code:

$$\mathcal{C} = \{\mathbf{c}_m\}_{m\in[M]} = \mathcal{C}_{\mathrm{LDPC}}, \qquad \text{so each } \mathbf{c}_m \text{ satisfies } H\mathbf{c}_m = \mathbf{0}. \tag{21}$$

Equivalently, each check row $j \in [P]$ enforces a single parity constraint

$$\sum_{\ell=0}^{L-1} H_{j,\ell}\,(\mathbf{c}_m)_\ell \;=\; 0 \qquad \text{in } \mathbb{F}_Q, \tag{22}$$

where only a small number of coefficients $H_{j,\ell}$ are nonzero. The defining feature of LDPC is that each constraint involves only a few symbol positions, enabling scalable constraint enforcement.

Tanner graph viewpoint (why sparsity matters). The matrix H induces a bipartite Tanner graph with variable nodes $\ell \in [L]$ (one per slot position) and check nodes $j \in [P]$ (one per parity equation). Define the edge set

$$E_H \triangleq \{(j,\ell) \in [P]\times[L] : H_{j,\ell} \neq 0\}, \qquad |E_H| = \#\{H_{j,\ell} \neq 0\}.$$

We also define neighbor sets

$$\mathcal{N}_{\mathrm{var}}(j) \triangleq \{\ell \in [L] : (j,\ell) \in E_H\}, \qquad \mathcal{N}_{\mathrm{chk}}(\ell) \triangleq \{j \in [P] : (j,\ell) \in E_H\}.$$

Then (22) can be written more explicitly as

$$\sum_{\ell' \in \mathcal{N}_{\mathrm{var}}(j)} H_{j,\ell'}\,(\mathbf{c}_m)_{\ell'} \;=\; 0 \qquad \text{in } \mathbb{F}_Q, \tag{23}$$

highlighting that each check touches only the few variables in $\mathcal{N}_{\mathrm{var}}(j)$.

Why we use LDPC in shared-codebook multiuser decoding. In URA, the symbol-wise soft detector outputs only *slot-wise* evidence for each slot ℓ and each symbol index $a \in [Q]$, but it does not specify how to assemble these local hypotheses into globally consistent messages. The LDPC constraint $Hc_m = 0$ provides strong global structure: among all Q^L possible length-L sequences, only those in $\mathcal{C}_{\mathrm{LDPC}}$ satisfy the parity checks. Thus, the multiuser decoder can use H to rule out spurious stitched sequences while still allowing efficient constraint propagation because H is sparse.

Soft information from the detector. Let $S \in \mathbb{R}^{L\times Q}$ be the evidence matrix produced by the fixed detector. A standard probabilistic form is the per-slot categorical belief

$$\lambda_\ell(a) \;\propto\; \exp(S_{\ell,a}), \qquad a \in [Q],\ \ell \in [L], \tag{24}$$

which can be interpreted (after normalization over a) as a soft belief that symbol a appeared in slot ℓ. LDPC decoding can be viewed as combining these local beliefs with the global parity constraints (20).

Standard LDPC decoding principle: belief propagation on the Tanner graph. A classical way to enforce (20) given soft beliefs (24) is belief propagation (BP) on the Tanner graph. BP passes Q-dimensional messages along edges $(j,\ell) \in E_H$. Denote variable-to-check messages by $m_{\ell\to j}(a)$ and check-to-variable messages by $m_{j\to\ell}(a)$, both defined for $a \in [Q]$. Here and below, the symbol arguments a and $\{a_{\ell'}\}$ are interpreted as elements of $\mathbb{F}_Q$ via the fixed labeling ϕ.

The variable update multiplies the local belief with incoming check messages:

$$m_{\ell\to j}(a) \;\propto\; \lambda_\ell(a) \prod_{j'\in\mathcal{N}_{\mathrm{chk}}(\ell)\setminus\{j\}} m_{j'\to\ell}(a). \tag{25}$$

The check update enforces the parity constraint by summing over assignments of neighboring variables that satisfy (23):

$$m_{j\to\ell}(a) \;\propto \sum_{\{a_{\ell'}\}_{\ell'\in\mathcal{N}_{\mathrm{var}}(j)\setminus\{\ell\}}} \mathbf{1}\left[H_{j,\ell}\, a \;+ \sum_{\ell'\in\mathcal{N}_{\mathrm{var}}(j)\setminus\{\ell\}} H_{j,\ell'}\, a_{\ell'} = 0\right] \prod_{\ell'\in\mathcal{N}_{\mathrm{var}}(j)\setminus\{\ell\}} m_{\ell'\to j}(a_{\ell'}). \tag{26}$$

After a fixed number of BP iterations, the approximate marginal at each variable node is

$$b_\ell(a) \;\propto\; \lambda_\ell(a) \prod_{j\in\mathcal{N}_{\mathrm{chk}}(\ell)} m_{j\to\ell}(a), \qquad a \in [Q], \tag{27}$$

and a hard decision can be made by $\hat{c}_\ell = \arg\max_{a\in[Q]} b_\ell(a)$. Our SIC-BP baseline uses this classical principle (with additional multiuser heuristics) to decode multiple rows.

How CIDER uses H. CIDER leverages the same parity structure but integrates it into the learned denoiser (Module B) through Tanner-graph propagation. In particular, the nonzero coefficients $\alpha = H_{j,\ell} \in \mathbb{F}_Q^\times$ define edge-dependent actions in $\mathbb{F}_Q$ under the fixed $[Q] \leftrightarrow \mathbb{F}_Q$ convention described above, and CIDER applies these coefficients consistently during propagation. In our implementation this is realized through fixed, field-consistent coefficient actions (e.g., T_α) inside the propagation module. This injects LDPC constraints while preserving the standard URA modularity (the symbol-wise soft detector and evidence interface S remain fixed).

Construction of H in our experiments. For each evaluation scale, we build the Tanner-graph connectivity pattern (i.e., the locations of nonzeros in H) from a structured regular graph that avoids short cycles, and then assign i.i.d. nonzero coefficients $\alpha_{j,\ell}$ on each edge:

$$\alpha_{j,\ell} = H_{j,\ell} \in \mathbb{F}_Q^\times \text{ i.i.d. for } (j,\ell) \in E_H, \qquad H_{j,\ell} = 0 \text{ for } (j,\ell) \notin E_H.$$

The nominal rate is approximately $R \approx (L-P)/L$ (and equals $(L-\mathrm{rank}(H))/L$ in general). For background on LDPC codes and belief-propagation decoding, see Gallager [1962].

G Additional background on masked discrete diffusion

This appendix collects standard identities for masked (absorbing-state) categorical diffusion, and summarizes the practical inference procedure used by masked denoising models (MDM).

G.1 Masked diffusion: reverse transitions

Time indexing and direction. We index refinement steps by $t \in \{0,1,\dots,T\}$, where $t=0$ denotes the most corrupted initialization (all-[Mask]) and $t=T$ denotes a fully revealed state. If one prefers the standard diffusion convention in which time increases with corruption, define $s \triangleq T-t$. Under this reparameterization, the forward noising direction is $s: 0 \to T$ (clean $\to$ corrupted) and the reverse denoising direction is $s: T \to 0$.

Discrete-step reverse model. At each refinement step t, the reverse process predicts clean tokens from the current masked state. Given denoiser prediction $\hat{x}_{\theta,j}(\tilde{X}^{(t)}, \xi, t) \in \Delta^Q$, unmasked positions are selected based on confidence ranking (see Section G.2).

Reverse transition. The reverse transition is parameterized by a denoiser that predicts a clean-token distribution from a noisy input and conditioning context ξ (in our URA instantiation, $\xi \equiv S$, where $S \in \mathbb{R}^{L\times Q}$ is the slot-wise evidence matrix from the fixed symbol-wise soft detector). Given a denoiser prediction $\hat{x}_{\theta,j}(\tilde{X}^{(t)}, \xi, t) \in \Delta^Q$, we define the reverse transition by substituting $\hat{x}_\theta$ into the analytic single-site posterior:

$$p_\theta\Big(\tilde{x}_j^{(s)} \mid \tilde{x}_j^{(t)}, \xi\Big) \triangleq q\Big(\tilde{x}_j^{(s)} \mid \tilde{x}_j^{(t)}, \hat{x}_{\theta,j}(\tilde{X}^{(t)}, \xi, t)\Big). \tag{28}$$

Here $s > t$ denotes a less-corrupted step than t under our $t = 0 \to T$ refinement convention. In practice, we use this identity only to motivate the standard masked-denoising objective and the iterative unmasking sampler; the implementation directly computes logits $\Lambda^{(t)}$ and updates the discrete masked state.

Uniform categorical corruption (D3PM-style). As a comparison point, instead of masked diffusion, D3PM replaces the absorbing mask with *uniform categorical* corruption [Austin et al., 2021]. Let $\bar{X}^{(t)} \in [Q]^{N_{\text{seq}}}$ denote the corrupted state (no mask token) and let $\bar{\gamma}_t \in [0,1]$ be the corruption rate. For each position $j \in [N_{\text{seq}}]$ and symbol $a \in [Q]$,

$$q_{\text{unif}}\Big(\bar{X}_j^{(t)} = a \,\Big|\, X_j = x_j\Big) = (1-\bar{\gamma}_t)\,\mathbf{1}[a = x_j] \;+\; \bar{\gamma}_t \cdot \frac{1}{Q}. \tag{29}$$

Equivalently, $\bar{X}_j^{(t)} = x_j$ with probability $(1-\bar{\gamma}_t)$, and otherwise $\bar{X}_j^{(t)}$ is resampled uniformly from $[Q]$.

G.2 Inference: confidence ranking and cosine reveal schedule

Unmasking order. While the masked diffusion model defines a family of reverse-time transitions, practical masked denoising models (MDM) must additionally specify *which sites are revealed at each iteration*. In particular, decoding maintains a discrete masked state $\tilde{X}^{(t)} \in ([Q] \cup \{[\mathsf{Mask}]\})^{K\times L}$ (with K rows and L slots) and iteratively replaces a subset of $[\mathsf{Mask}]$ entries with predicted symbols over T refinement steps. Thus, inference is determined not only by the denoiser outputs but also by an *unmasking schedule*.

We adopt a cosine reveal schedule [Chang et al., 2022] to determine how many sites should be unmasked by step t:

$$\rho(t) \triangleq \frac{1}{2}\left(1 - \cos\left(\pi \frac{t}{T}\right)\right), \qquad t = 0, 1, \ldots, T. \tag{30}$$

During inference, the scalar mask-ratio input to the denoiser is set to the remaining masked fraction,

$$\gamma_t^{\text{infer}} = \frac{\big|\{(k,\ell) : \tilde{X}_{k,\ell}^{(t-1)} = [\mathsf{Mask}]\}\big|}{KL},$$

so the conditioning matches the input actually seen at step t.

Concretely, at step t we compute logits

$$\Lambda^{(t)} = f_\theta(\tilde{X}^{(t-1)}, S, \gamma_t^{\text{infer}})$$

and probabilities

$$p_{k,\ell,:}^{(t)} = \text{softmax}\Big(\Lambda_{k,\ell,:}^{(t)} / \tau_{\text{infer},t}\Big).$$

We define the per-site *confidence* as

$$\kappa_{k,\ell}^{(t)} \;\triangleq\; \max_{a\in[Q]} p_{k,\ell,a}^{(t)},$$

to avoid clashing with the codebook notation c_m in the main text.

At step t, we reveal the highest-confidence entries among the currently masked sites so that the *cumulative* fraction of revealed sites is approximately $\rho(t)$.

First-reveal stabilization. In our default CIDER inference procedure, we use a first-reveal stabilization rule that gives priority to row hypotheses without any revealed site. For $K \geq 3$, whenever a top-ranked masked site belongs to such a row k, we reveal only that single site at that step, which anchors row k. Once every row is anchored (and after the remasking in PRISM passes), decoding follows the cosine schedule in (30). This keeps early reveals from concentrating on a few rows while others remain fully masked, which matters most at higher per-bin loads. For $K \leq 2$, the rule reduces to first revealing the most confident site of one randomly chosen row.

G.3 PRISM-style quality head and remasking plug-in

This section describes how we plug a PRISM-style confidence head into CIDER and use it for quality-guided remasking at inference [Kim et al., 2026]. The key property is that the head is trained post-hoc and does not modify the backbone denoiser; it only reads the backbone latent representations and outputs per-token correctness probabilities that we use to trigger selective remasking.

Backbone outputs and notation. CIDER produces token logits $\Lambda^{(t)} \in \mathbb{R}^{K\times L\times Q}$ at each refinement step $t \in \{1,\dots,T\}$. We denote the refined per-step latent used by the confidence head by $\hat{Z}^{(t)} \in \mathbb{R}^{K\times L\times D}$. We write

$$p^{(t)}_{k,\ell,:} \triangleq \mathrm{softmax}\Big(\Lambda^{(t)}_{k,\ell,:}/\tau_t\Big) \in \Delta^Q$$

for the categorical distribution at site (k,ℓ) (with temperature $\tau_t > 0$ if used). Let $X^\star \in [Q]^{K\times L}$ be the ground-truth codeword grid (row order immaterial). As in the main text, we use Hungarian matching to align predicted rows to ground-truth rows; we denote the resulting permutation by $\pi^\star \in \mathfrak{S}_K$.

Per-token quality head. We attach a lightweight two-layer MLP g_ψ to each site latent and predict a per-token correctness probability

$$\omega^{(t)}_{k,\ell} \;=\; \sigma\Big(g_\psi\Big(\hat{Z}^{(t)}_{k,\ell}\Big)\Big) \in [0,1], \tag{31}$$

where $\sigma(\cdot)$ is the sigmoid. Concretely, g_ψ is a 2-layer MLP (e.g., LayerNorm/GELU/Dropout between layers) mapping $\mathbb{R}^D \to \mathbb{R}$.

PRISM-style supervision for the head. We train the head using PRISM-style labels constructed from the backbone's own sampled predictions. For each training example, we sample a diffusion step (mask ratio) and form a partially-masked grid input $\tilde{X}^{(t)}$, run the frozen backbone to obtain logits $\Lambda^{(t)}$ and probabilities $p^{(t)}_{k,\ell,:}$ at masked sites, and compute the row-matching permutation $\pi^\star$ (as in the main text) using a discrete grid derived from $\Lambda^{(t)}$ (e.g., per-site argmax). For a selected subset of masked sites $\mathcal{S} \subseteq \{(k,\ell) : \tilde{X}^{(t)}_{k,\ell} = [\mathsf{Mask}]\}$, we sample a token prediction

$$\hat{x}^{(t)}_{k,\ell} \sim \mathrm{Cat}\Big(p^{(t)}_{k,\ell,:}\Big), \qquad (k,\ell) \in \mathcal{S}. \tag{32}$$

We define the binary correctness label (using the matched ground-truth row)

$$b^{(t)}_{k,\ell} \;\triangleq\; \mathbf{1}\Big[\hat{x}^{(t)}_{k,\ell} = X^\star_{\pi^\star(k),\ell}\Big], \tag{33}$$

and minimize binary cross-entropy on $\mathcal{S}$:

$$\mathcal{L}_{\mathrm{qual}}(\psi) \;=\; \sum_{(k,\ell)\in\mathcal{S}} \Big(- b^{(t)}_{k,\ell} \log \omega^{(t)}_{k,\ell} - (1 - b^{(t)}_{k,\ell}) \log(1 - \omega^{(t)}_{k,\ell})\Big). \tag{34}$$

In all experiments, the backbone denoiser is frozen when training g_ψ; only the head parameters ψ are updated.

Sequence-level confidence for multiuser decoding. We make remasking decisions at the *sequence* (row) level. After a decoding pass completes (all tokens revealed at the final step $t = T$), we compute a sequence-level confidence score by averaging per-token quality predictions across slots:

$$\bar{\omega}_k \;\triangleq\; \frac{1}{L}\sum_{\ell=0}^{L-1} \omega^{(T)}_{k,\ell}. \tag{35}$$

We interpret smaller $\bar{\omega}_k$ as indicating a more error-prone sequence (row) that is likely to contain incorrect symbols across rows.

Quality-guided remasking and second pass. Let $\hat{X} \in [Q]^{K\times L}$ be the discrete decoded grid after the first pass. We use a multi-stage threshold-based remasking strategy indexed by $s \in \{1, \dots, \varrho\}$, with quality thresholds $\{\varphi_s\}_{s=1}^{\varrho}$.

After each decoding pass, we identify low-confidence sequences as those falling below the threshold:

$$\mathcal{K}_{\text{low}} \triangleq \{k \in [K] : \bar{\omega}_k < \varphi_s\}.$$

We initialize the second-pass discrete state by remasking exactly these sequences:

$$X^{(0)}_{\text{rm}\,k,\ell} = \begin{cases} [\mathsf{Mask}], & k \in \mathcal{K}_{\text{low}}, \\ \hat{X}_{k,\ell}, & k \notin \mathcal{K}_{\text{low}}. \end{cases} \tag{36}$$

We then rerun the masked-diffusion inference procedure starting from $X^{(0)}_{\text{rm}}$, while clamping the high-confidence codewords throughout the second pass: for all refinement steps t and all $k \notin \mathcal{K}_{\text{low}}$, we enforce $X^{(t)}_{\text{rm}\,k,\ell} = \hat{X}_{k,\ell}$ (i.e., token updates are disallowed outside $\mathcal{K}_{\text{low}}$). The number of refinement steps for each remasking pass scales with the number of sequences being remasked:

$$T_{\text{rm}} = \max\left(1,\ \left\lfloor T \cdot \frac{|\mathcal{K}_{\text{low}}|}{K} \right\rfloor\right)$$

where T is the number of steps used in the initial decoding pass. This process repeats (increasing s) until $\mathcal{K}_{\text{low}} = \emptyset$ or all ϱ thresholds are exhausted.

Algorithmic summary. Algorithm 1 summarizes the procedure for one remasking round.

Algorithm 1 PRISM-style quality-guided remasking for CIDER

1. Run masked-diffusion decoding for T steps to obtain $\hat{X}$ and final refined latents $\hat{Z}^{(T)}$.
2. Compute per-token qualities $\omega^{(T)}_{k,\ell} = \sigma(g_\psi(\hat{Z}^{(T)}_{k,\ell}))$ and sequence confidences $\bar{\omega}_k = \frac{1}{L}\sum_\ell \omega^{(T)}_{k,\ell}$.
3. Select $\mathcal{K}_{\text{low}}$ as the sequences with $\bar{\omega}_k < \varphi_s$.
4. Initialize $X^{(0)}_{\text{rm}}$ by remasking sequences (rows) in $\mathcal{K}_{\text{low}}$ (Eq. (36)).
5. Rerun masked-diffusion decoding from $X^{(0)}_{\text{rm}}$ for T_{rm} steps while clamping $k \notin \mathcal{K}_{\text{low}}$; output the refined grid.

Practical notes. (i) The remasking head adds negligible runtime because it is a small per-token MLP applied to already-computed latents. (ii) The number of thresholds ϱ and their values trade compute for reliability; more stages with stricter (higher) thresholds trigger additional remasking passes but improve accuracy. (iii) We use $\varrho = 1, 2, 3$ remasking rounds for $K = 6, 7, 8$, respectively, in our reported results.

H Experimental setup and implementation details

This appendix collects implementation details omitted from Section 5: (i) data generation and channel model, (ii) sensing matrices and symbol-wise soft detector evidence construction, (iii) multiuser code families, (iv) training and inference schedules, and (v) per-scale model hyperparameters.

H.1 Simulation pipeline and channel model

For each scale in Table 6, we generate synthetic frames by sampling K active messages uniformly from $[M]$, mapping them through the shared codebook to obtain the ground-truth grid $X^\star \in [Q]^{K\times L}$, and transmitting one signature index per slot. For each slot $\ell \in [L]$, we form the sparse activity vector $U^{(\ell)}$ as in Equation (10) and generate

$$Y^{(\ell)} = A^{(\ell)}U^{(\ell)} + \epsilon^{(\ell)}, \qquad \epsilon^{(\ell)} \sim \mathcal{CN}(0, \sigma^2 I). \tag{37}$$

Simulation constants. Unless stated otherwise, we use $K = 2$ active users. We generate $Y^{(\ell)} = A^{(\ell)}U^{(\ell)} + \epsilon^{(\ell)}$ with $\epsilon^{(\ell)} \sim \mathcal{CN}(0, \sigma^2 I)$ and fixed $\sigma^2 = 1.0$. We control the effective SNR by scaling the transmit power in the symbol encoder as $P_{\text{sym}} = \frac{B \cdot 10^{E_b/10}}{L}$ (with E_b in dB).

Dataset sizes and sensing dimensions. We use $n_s = 24$ in our main implementation unless otherwise specified. Additionally, we use a $70\text{K}/15\text{K}/15\text{K}$ train/validation/test split per scale.

H.2 Sensing-matrix construction

All experiments use the partial-DFT sensing matrix. Let $\mathbf{F} \in \mathbb{C}^{Q \times Q}$ be the unnormalized DFT matrix ($\mathbf{F}_{k,n} = e^{-2\pi i k n/Q}$) and let $\mathcal{R} \subset \{0, \ldots, Q-1\}$ be n_s row indices drawn uniformly at random:

$$\mathbf{A}_{\text{DFT}} = \frac{1}{\sqrt{n_s}} \mathbf{F}_{\mathcal{R},:} \in \mathbb{C}^{n_s \times Q}. \tag{38}$$

This normalization ensures each column a_q satisfies $\|a_q\|_2^2 = 1$.

H.3 Per-user per-channel-use SNR

Since each column of the sensing matrix is normalized, one transmitted symbol carries energy P_{sym} spread uniformly across its n_s complex channel uses. Thus the per-user per-channel-use signal energy is P_{sym}/n_s, and

$$\text{SNR} = \frac{P_{\text{sym}}}{n_s \, \sigma^2} = \frac{B \cdot 10^{E_b/10}}{L \, n_s \, \sigma^2}, \tag{39}$$

$$\text{SNR}\,[\text{dB}] = E_b - 10 \log_{10}\big(L n_s / B\big) \quad (\sigma^2 = 1). \tag{40}$$

H.4 Fixed symbol-wise soft detector and evidence interface

For each slot ℓ, the fixed symbol-wise soft detector (AMP–MMSE) takes $(Y^{(\ell)}, A^{(\ell)})$ and produces log-posterior evidence scores $s_{\ell,a}$ for every $a \in [Q]$. We aggregate these into the evidence heatmap $S \in \mathbb{R}^{L \times Q}$ with entries $S_{\ell,a} = s_{\ell,a}$. The matrices S are pre-computed offline and stored in the dataset; all multiuser decoders operate only on S.

H.5 Multiuser-code families and parity-check metadata

In the experiments reported in this paper, we use LDPC codes (with rate R as specified per experiment). We store the parity-check matrix $H \in \mathbb{F}_Q^{P \times L}$ and provide it to parity-aware decoders.

H.6 Training and inference schedules

Optimization. All learned models are trained with AdamW under identical dataset splits (shared across model families unless otherwise stated).

Permutation-invariant training. Because the K decoded rows are unordered, we align predicted rows to target rows during training. For a masked training example, let $p^{(t)}_{k,\ell,:}$ be the predicted categorical distribution at site (k, ℓ), $\Omega_t \subseteq [K] \times [L]$ be the masked grid sites, $\Omega_{t,k} = \{\ell : (k, \ell) \in \Omega_t\}$ the masked slots of row k, and $\mathcal{V}_t = \{\ell : \exists k,\ (k, \ell) \notin \Omega_t\}$ the slots unmasked in at least one row (all slots if $\mathcal{V}_t = \emptyset$). We form the row-matching cost

$$D_{k,k'} = \frac{1}{|\mathcal{V}_t|} \sum_{\ell \in \mathcal{V}_t} \text{CE}\Big(X^{\star}_{k',\ell}, p^{(t)}_{k,\ell,:}\Big),$$

compute

$$\pi^{\star} = \arg\min_{\pi \in \mathfrak{S}_K} \sum_{k=0}^{K-1} D_{k,\pi(k)},$$

and minimize the matched cross-entropy on masked sites, averaged within each row and then over rows,

$$\mathcal{L}_{\text{train}} = \frac{1}{K} \sum_{k:\, \Omega_{t,k} \neq \emptyset} \frac{1}{|\Omega_{t,k}|} \sum_{\ell \in \Omega_{t,k}} \mathrm{CE}\Big(X^{\star}_{\pi^{\star}(k),\ell}, p^{(t)}_{k,\ell,:}\Big),$$

averaged over the minibatch. This matching removes the arbitrary row order from the supervision while preserving the standard masked-diffusion denoising objective.

MDM mask-ratio sampling (training). We sample a discrete timestep t uniformly from $\{0, 1, \ldots, T_{\text{train}}\}$ with T_{train} to each model's step count (10/12/20/24 for Tiny/Small/Moderate/Large) and set

$$\gamma_t \triangleq \gamma_{\max} - (\gamma_{\max} - \gamma_{\min}) \frac{t}{T_{\text{train}}}, \qquad (\gamma_{\min}, \gamma_{\max}) = (0.1, 1.0). \tag{41}$$

We use a short warmup: for the first 4 epochs we exclude $t = 0$ (fully masked), and afterwards we include $t = 0$ (so fully masked examples occur with probability $1/(T_{\text{train}} + 1)$ under uniform t sampling).

MDM inference schedule (cosine reveal). We run T refinement steps (scale-dependent; see Table 7) and reveal tokens by confidence ranking according to the cosine schedule in (30).

Inference temperature annealing. For $K = 2$, confidences used for reveal ranking are computed with a cosine-annealed temperature that increases from $\tau_{\min}$ at the first refinement step to $\tau_{\max}$ at the final step:

$$\tau_{\text{infer},t} \triangleq \tau_{\min}\, w(t) + \tau_{\max}\, \big(1 - w(t)\big), \qquad w(t) \triangleq \frac{1}{2}\left(1 + \cos\left(\pi \frac{t-1}{T-1}\right)\right). \tag{42}$$

The denoiser probabilities are computed as $p^{(t)}_{k,\ell,:} = \mathrm{softmax}(\Lambda^{(t)}_{k,\ell,:}/\tau_{\text{infer},t})$. The $K = 3$–5 decoder anneals in the opposite direction, from $\tau_{\max}$ at the first step to $\tau_{\min}$ at the last, and the PRISM decoder ($K \geq 6$) uses the fixed temperature throughout.

H.7 Model sizes by evaluation scale

We report four evaluation scales (Tiny/Small/Moderate/Large) in Table 7 with a fixed alphabet size $Q = 64$ and varying codeword lengths $L \in \{12, 18, 24, 48\}$. For each scale, we scale the denoiser capacity (D, layers/heads) and compare the same set of baselines under an identical evidence interface S and Hungarian-matched supervision.

I Complexity derivation

Let J be the per-slot candidate list size retained from S (top-J indices per slot). Let $H \in \mathbb{F}_Q^{P \times L}$ be the parity-check matrix and let $|E_H|$ be the number of nonzeros in H (i.e., Tanner-graph edges). We write I_{BP} for the number of Tanner-graph message-passing iterations performed by SIC-BP. For CIDER, we write T for the number of refinement steps and N_{layer} for the number of stacked denoiser blocks inside one refinement step.

Top-J exhaustive search. Top-J Exhaustive search forms candidate codewords by selecting one of J candidates per slot, yielding J^L candidates in the worst case. Checking parity validity for one candidate can be implemented by aggregating along Tanner edges, costing $\mathcal{O}(|E_H|)$. Thus the worst-case complexity is

$$\mathcal{O}(J^L) \times \mathcal{O}(|E_H|) = \mathcal{O}(J^L |E_H|),$$

which is exponential in L (over the reduced Top-J search space).

SIC-BP. SIC-BP performs belief propagation (BP) on the non-binary Tanner graph. The dominant operation in non-binary BP is the check-to-variable update. For a check node of degree d_c, computing one outgoing check-to-variable message amounts to convolving $(d_c - 1)$ distributions over $\mathbb{F}_Q$. With direct GF(Q) convolution, one convolution costs $\mathcal{O}(Q^2)$, so one outgoing message costs $\mathcal{O}(d_c\, Q^2)$. Since LDPC degrees are constant in our setting, we treat d_c as a constant factor and write the per-edge BP update cost as $\Theta(Q^2)$. Therefore, I_{BP} BP iterations over K decoded rows cost

$$\mathcal{O}\big(K\, I_{\mathrm{BP}}\, |E_H|\, Q^2\big)\,.$$

FFT-BP. FFT-BP is algorithmically identical to SIC-BP but accelerates the check-to-variable update by exploiting the additive structure of $\mathbb{F}_{2^m}$. Since field addition coincides with bitwise XOR, the convolution of (d_c-1) distributions over $\mathbb{F}_Q$ becomes pointwise multiplication in the Walsh–Hadamard transform (WHT) domain. Each WHT and its inverse costs $\mathcal{O}(Q \log Q)$, so the per-edge BP update cost drops from $\Theta(Q^2)$ to $\Theta(Q \log Q)$. This WHT-based formulation is the one used by recent coded random-access work [Ebert et al., 2022]. At $Q = 64$ this reduces the per-edge operation count by about $10\times$, which matches the wall-clock speedup in per-sample CPU implementation.

CIDER. CIDER runs T refinement steps on the discrete grid $X^{(t)}$ and constructs a latent grid of size $K \times L \times D$ at each step. We summarize the dominant costs per refinement step.

(i) Logit projection and demixing/evidence fusion. Computing Q logits at each of KL sites from D-dimensional latents costs $\mathcal{O}(KLQD)$, and the demixing/evidence fusion has the same Q-summation structure per site. Applying N_{layer} denoiser blocks multiplies this term by N_{layer}.

(ii) Parity-aware propagation. Parity propagation processes Tanner edges for each of the K rows. The main per-edge operation is the coefficient action used for normalization and denormalization inside Module B. In our implementation, a coefficient action is applied as

$$T_\alpha(x) \triangleq W^\top \Pi_\alpha W x,$$

which can be implemented by projecting to symbol space, applying the fixed permutation, and projecting back. This costs $\mathcal{O}(QD)$ per application. Since normalization and denormalization are applied on each edge (constant number of times per edge per block), the parity-propagation cost scales as

$$\mathcal{O}(N_{\mathrm{layer}}\, K\, |E_H|\, QD)\,,$$

up to constant factors from local aggregation over a constant check degree.

(iii) Stabilization (optional). If enabled, applying Φ_U over all KL sites costs $\mathcal{O}(KL \cdot \mathrm{cost}(\Phi_U))$ per refinement step.

Putting these together, the total cost over T refinement steps is

$$\mathcal{O}\big(T\, N_{\mathrm{layer}}\big[KL(QD + D^2) + K|E_H|(QD + D^2)\big] + T\, KL \cdot \mathrm{cost}(\Phi_U)\big)\,.$$

Reading the comparison. This asymptotic accounting should be interpreted together with hardware execution. CIDER is not claimed to have a smaller scalar operation count than every optimized BP variant in all regimes; for example, FFT-BP reduces non-binary check updates to $\mathcal{O}(Q \log Q)$. The empirical speedup comes from fixed-depth, highly parallel dense tensor operations on GPU, avoidance of sequential SIC-style peeling, and avoidance of Top-J combinatorial enumeration.

J Exact configurations for reproducibility

This appendix summarizes the exact configuration values used in CIDER experiments (dataset scales, model sizing, and training/inference hyperparameters).

J.1 Dataset scales (LDPC over GF(64))

All scales use $Q = 64$ (GF(64)) and $K = 2$ active users. We vary the codeword length L while keeping the code rate fixed at $R = 1/3$. To avoid notational conflicts with the main text (where k denotes a row index), we denote the number of parity checks by P and the number of information symbols by $L_{\mathrm{info}} \triangleq L - P$.

Table 6: LDPC dataset scales used in all experiments (GF(64), $K = 2$).

Scale	Q	L	P	L_{info}	$R = L_{\text{info}}/L$
Tiny	64	12	8	4	1/3
Small	64	18	12	6	1/3
Moderate	64	24	16	8	1/3
Large	64	48	32	16	1/3

Information payload. Each symbol carries $\log_2(64) = 6$ bits, so the payload size is $6L_{\text{info}}$ bits per message.

J.2 Model sizing and inference steps

Diffusion models run in discrete masked-token mode (`use_soft_input=false`) and use scale-dependent model sizing and inference steps.

Table 7: Diffusion-model sizing by scale.

Scale	D	N_{layer}	#Heads	Inference steps T
Tiny	128	4	4	12
Small	128	6	4	16
Moderate	128	6	4	20
Large	128	8	4	28

J.3 Per-K model sizing

Diffusion models run in discrete masked-token mode (`use_soft_input=false`) and use scale-dependent model sizing, inference steps, and remasking threshold for different K.

Table 8: Diffusion-model sizing per K. For $K \geq 6$, the PRISM-head pipeline performs ϱ remasking rounds on top of the initial T_{init}-step diffusion pass, gated by the threshold schedule $(\varphi_1, \ldots, \varphi_\varrho)$.

K	D	N_{layer}	#Heads	T_{init}	Remask rounds ϱ	Thresholds $(\varphi_1, \varphi_2, \varphi_3)$
2	128	4	4	12	–	–
3	128	4	4	24	–	–
4	128	4	4	42	–	–
5	128	4	4	60	–	–
6	128	5	4	50	1	(0.97)
7	128	5	4	62	2	$(0.96,\ 0.90)$
8	128	5	4	74	3	$(0.96,\ 0.90,\ 0.85)$

J.4 Training hyperparameters

Table 9: Training hyperparameters.

Parameter	Value
Batch size	128
Epochs	100
Optimizer	AdamW (weight decay 10^{-4} for CNN/MLP/Transformer/GNN/NBP weight decay 10^{-2} for CIDER/TA/MDD)
Learning rate	10^{-3}
LR schedule	cosine, warmup 10 epochs, $\eta_{\min} = 10^{-6}$
EMA decay	0.9999
Precision	16-mixed
Mask ratio γ	γ_t with $t \sim \mathrm{Unif}\{0, \ldots, T_{\mathrm{train}}\}$, $\gamma_t \in [0.1, 1.0]$ after warmup
Mask warmup	4 epochs (exclude $\gamma = 1$)
$\Pr[\gamma = 1.0]$	$1/(T_{\mathrm{train}} + 1)$ after warmup
AMP iterations I_{AMP}	10

J.5 Compute resources

Training and inference were run on separate machines.

Training environment. All model training used a single NVIDIA A100-SXM4-80GB GPU operated in MIG mode with a 40GB compute slice. Each training run used a single GPU slice; no multi-GPU or distributed training was used.

Training cost. Backbone training at $K = 2$ (100 epochs, batch size 128, single GPU) took $\sim$4.75 h at the tiny scale, $\sim$4.9 h at small, $\sim$5.0 h at moderate and $\sim$6.4 h at large. Backbone training becomes more expensive at higher K because the Hungarian-matching step in the loss is $O(K^3)$ per sample.

Inference / evaluation environment. All evaluation results reported in this paper—including neural and classical baselines—were generated on a workstation with a single NVIDIA GeForce RTX 3090 GPU with 24GB memory and an Intel Core i5-14500 CPU.

K Architectures of compared multiuser decoders

K.1 Shared input/output interfaces

Evidence and outputs. All compared decoders consume the same symbol-wise soft detector evidence $S \in \mathbb{R}^{L \times Q}$ (batched as $\mathbb{R}^{B_{\mathrm{sz}} \times L \times Q}$) and output token logits $\Lambda \in \mathbb{R}^{K \times L \times Q}$ (or batched as $\mathbb{R}^{B_{\mathrm{sz}} \times K \times L \times Q}$). The decoded grid is obtained by per-site $\arg\max$ on Λ. When using a batch dimension, we denote the batch size by B_{sz} to avoid confusion with the payload length B in the main text. Architecture diagrams are provided in Figures 6 and 7. All accuracy metrics are computed after Hungarian row matching to account for permutation invariance.

K-head outputs. For one-shot baselines, a shared length-L hidden sequence is produced and then mapped to K output rows using K independent heads, each predicting a distribution over $[Q]$ per slot.

K.2 MLP baseline: direct flattening of evidence

Figure 6 illustrates this baseline.

Tokenization/embedding. The MLP baseline applies no learned token embedding. It flattens the evidence as $x = \mathrm{vec}(S) \in \mathbb{R}^{B_{\mathrm{sz}} \times (LQ)}$ and feeds x into an MLP.

Backbone and output. The backbone is an n_{mlp}-layer MLP of hidden width `hidden_dim`. Each hidden layer uses an affine transform followed by LayerNorm, GELU, and dropout (rate p), and the final layer projects to logits $\Lambda \in \mathbb{R}^{B_{\mathrm{sz}} \times K \times L \times Q}$.

K.3 CNN baseline: Conv1d along the slot axis

Figure 6 illustrates this baseline.

Input projection. The CNN baseline first projects each slot evidence vector $S_{\ell,:} \in \mathbb{R}^Q$ to a C-dimensional feature via Linear($Q \to C$)→LayerNorm→GELU. Convolutions are applied only along the slot axis L (no convolution across Q).

Convolution axis (key point). The backbone is a stack of residual 1D convolution blocks operating over the slot dimension L (padding preserves length).

Residual blocks. Each residual block uses *two* Conv1d layers:

$$\begin{aligned} &\text{res} \leftarrow x, \\ &\quad x \leftarrow \text{Conv1d} \to \text{BN} \to \text{GELU} \to \text{Dropout}, \\ &\quad x \leftarrow \text{Conv1d} \to \text{BN}, \\ &\quad x \leftarrow \text{GELU}(x + \text{res}). \end{aligned}$$

Output heads. After the backbone, K independent slot-wise heads map features to logits via Linear($C \to C$)→GELU→Dropout→Linear($C \to Q$), yielding $\Lambda \in \mathbb{R}^{B_{\mathrm{sz}} \times K \times L \times Q}$.

K.4 Transformer baseline: linear projection + sinusoidal positional encoding

Figure 6 illustrates this baseline.

Evidence tokenization. The transformer treats each slot as a token by applying a learned linear projection $Q \to D_{\mathrm{tr}}$, producing $x \in \mathbb{R}^{B_{\mathrm{sz}} \times L \times D_{\mathrm{tr}}}$.

Positional encoding and backbone. Sinusoidal positional encodings are added (with dropout rate p), followed by a standard Transformer encoder over the L slot tokens. Each block uses multi-head self-attention and a position-wise FFN with GELU, both wrapped with residual connections and LayerNorm.

Output. The model uses K independent slot-wise output heads, each implemented as Linear($D_{\mathrm{tr}} \to D_{\mathrm{tr}}$)→GELU→Dropout→Linear($D_{\mathrm{tr}} \to Q$), and stacks the results to obtain $\Lambda \in \mathbb{R}^{B_{\mathrm{sz}} \times K \times L \times Q}$.

K.5 GNN baseline

Figure 6 illustrates this baseline.

Evidence tokenization. The GNN baseline treats each variable node as a token by applying a learned linear projection $Q \to D$, producing $x \in \mathbb{R}^{B_{\mathrm{sz}} \times L \times D_{\mathrm{gnn}}}$. Learnable row embeddings $s_k \in \mathbb{R}^D_{\mathrm{gnn}}$ are added to create K parallel copies.

Message passing on H. Each GNN layer performs bidirectional message passing on the Tanner graph defined by H. Variable nodes aggregate to check nodes via H, check nodes update their state, then send messages back via $H^\top$. Check node states are initialized from a learnable parameter. All transformations use two-layer MLPs (LN→Linear→GELU→Dropout→Linear) and residual connections.

Output heads. The model uses K independent output heads, each implemented as Linear($D_{\mathrm{gnn}} \to D_{\mathrm{gnn}}$)→GELU→Dropout→Linear($D_{\mathrm{gnn}} \to Q$), producing $\Lambda \in \mathbb{R}^{B_{\mathrm{sz}} \times K \times L \times Q}$.

K.6 Neural Belief Propagation (NBP) baseline

Figure 6 illustrates this baseline.

Evidence tokenization. A learned projection $Q \to D_{\text{nbp}}$ encodes channel evidence to $x \in \mathbb{R}^{B_{\text{sz}} \times L \times D_{\text{nbp}}}$, with row embeddings added for K copies.

Message passing on H. Each layer unfolds one BP iteration on the Tanner graph. Messages flow through VN→CN through H and CN→VN through $H^\top$, with 2-layer MLP at each step. Two key differences from GNN: (1) *channel skip*—the encoded evidence x is re-injected at every layer rather than only at initialization; (2) *learnable damping*—beliefs are updated as $b^{(\ell)} = \alpha b^{(\ell-1)} + (1-\alpha)\tilde{b}$ where $\alpha = \sigma(\theta)$ is learned.

Output heads. The model uses K independent output heads, each implemented as Linear($D_{\text{nbp}} \to D_{\text{nbp}}$)→GELU→Dropout→Linear($D_{\text{nbp}} \to Q$), producing $\Lambda \in \mathbb{R}^{B_{\text{sz}} \times K \times L \times Q}$.

K.7 Tanner-Attention (TA) baseline

Figure 6 illustrates this baseline.

Overview. TA is a one-shot, parity-aware baseline that injects code constraints via attention/message passing on the Tanner graph. Given evidence $S \in \mathbb{R}^{L \times Q}$ (mini-batched as $\mathbb{R}^{B_{\text{sz}} \times L \times Q}$) and a sparse parity-check matrix $H \in \mathbb{F}_Q^{P \times L}$, TA constructs variable-node tokens (one per slot) and check-node tokens (one per parity equation), then performs neighbor-restricted attention updates along Tanner edges. TA explicitly exploits Tanner sparsity by restricting interactions to graph neighborhoods. It is not belief propagation: it performs a single forward pass of Tanner-structured attention/message passing.

Tokenization and shared shapes. TA represents (i) variable nodes as a length-L sequence of embeddings in $\mathbb{R}^{B_{\text{sz}} \times L \times D}$ and (ii) check nodes as a length-P sequence in $\mathbb{R}^{B_{\text{sz}} \times P \times D}$. The output is a logit tensor $\Lambda \in \mathbb{R}^{B_{\text{sz}} \times K \times L \times Q}$, obtained by projecting the final variable-node embeddings through K independent output heads.

Evidence encoder. Evidence vectors $S_{\ell,:} \in \mathbb{R}^Q$ are converted into variable-node inputs by a small MLP encoder: LayerNorm → Linear → GELU → Dropout → Linear, producing

$$\mathbf{T}^{\text{var},(0)} \in \mathbb{R}^{B_{\text{sz}} \times L \times D}.$$

A positional embedding over slots is added to break slot symmetry (followed by dropout), yielding the initial variable-node tokens.

Check-node initialization. Check-node tokens are initialized by a learnable embedding indexed by check equation $j \in [P]$ (and broadcast across the batch), producing an initial sequence

$$\mathbf{T}^{\text{chk},(0)} \in \mathbb{R}^{B_{\text{sz}} \times P \times D}.$$

This makes TA "structure-aware" in the sense that the graph topology comes from H, while the check-node states are learned.

Tanner-structured message passing. Let $\mathcal{N}_{\text{var}}(j) = \{\ell \in [L] \,:\, H_{j,\ell} \neq 0\}$ denote the variable neighbors of check j, and $\mathcal{N}_{\text{chk}}(\ell) = \{j \in [P] \,:\, H_{j,\ell} \neq 0\}$ denote check neighbors of variable ℓ. TA alternates two neighbor-restricted updates:

Variable → check (VN→CN). Each check token aggregates messages from its neighboring variables using multi-head attention where queries are $\mathbf{T}^{\text{chk}}$ and keys/values are $\mathbf{T}^{\text{var}}$ restricted to $\mathcal{N}_{\text{var}}(j)$. The update uses residual connections and LayerNorm, followed by a position-wise FFN (GELU, dropout).

Check → variable (CN→VN). Each variable token aggregates messages from its neighboring checks using attention where queries are variable tokens and keys/values are the check tokens restricted to $\mathcal{N}_{\text{chk}}(\ell)$, again followed by residual + LayerNorm and an FFN (GELU, dropout).

Non-binary coefficient handling. For non-binary LDPC codes, each edge (j, ℓ) carries a coefficient $H_{j,\ell} \in \mathbb{F}_Q^\times$. TA accounts for coefficients by applying a coefficient-conditioned transform on edge messages before aggregation, so that check-node updates depend on coefficient-normalized neighbor information and variable-node updates receive denormalized messages. (Implementation-wise, this is realized by fixed, coefficient-indexed mixing/permutation operators consistent with the $\mathbb{F}_Q$ convention of H.)

Output heads. After L_{mp} message-passing layers, the final variable tokens $v \in \mathbb{R}^{B_{\mathrm{sz}} \times L \times D}$ are mapped to logits via K independent heads (each a small MLP head with GELU and dropout, ending in a Linear to Q logits), stacked to form $\Lambda \in \mathbb{R}^{B_{\mathrm{sz}} \times K \times L \times Q}$. Decoding uses per-site argmax.

Algorithm (forward pass). Given a mini-batch evidence tensor $S \in \mathbb{R}^{B_{\mathrm{sz}} \times L \times Q}$ and parity matrix $H \in \mathbb{F}_Q^{P \times L}$, TA computes logits $\Lambda \in \mathbb{R}^{B_{\mathrm{sz}} \times K \times L \times Q}$ as:

1. **Encode evidence.** Apply an evidence MLP (LayerNorm, GELU, dropout) to obtain initial variable tokens $\mathbf{T}^{\mathrm{var},(0)}$, and add slot positional embeddings.
2. **Initialize checks.** Create check tokens $\mathbf{T}^{\mathrm{chk},(0)}$ from a learnable check-index embedding.
3. **Repeat for L_{mp} layers:**
   (a) *VN→CN update:* update each check token by neighbor-restricted attention over $\{\mathbf{T}_\ell^{\mathrm{var}} : \ell \in \mathcal{N}_{\mathrm{var}}(j)\}$, applying coefficient conditioning on edges (j, ℓ); apply residual + LayerNorm and an FFN.
   (b) *CN→VN update:* update each variable token by neighbor-restricted attention over $\{\mathbf{T}_j^{\mathrm{chk}} : j \in \mathcal{N}_{\mathrm{chk}}(\ell)\}$, again with coefficient conditioning; apply residual + LayerNorm and an FFN.
4. **Project to logits.** Apply K independent output heads on variable tokens and stack outputs to obtain Λ.

Implementation notes. (i) All attention/FFN blocks follow the standard Transformer pattern (multi-head attention, residual connections, LayerNorm, dropout; FFN uses GELU). (ii) This baseline is *one-shot*: unlike MDD/CIDER, it does not perform iterative diffusion refinement; all parity conditioning is injected in a single forward pass via Tanner-structured updates.

K.8 Masked Diffusion Decoder (MDD) baseline: concatenation fusion of discrete tokens and evidence

Figure 7 compares the MDD denoiser with CIDER under the same masked-refinement loop.

Overview. MDD instantiates masked discrete diffusion with a generic Transformer denoiser. At each refinement step, the denoiser consumes a partially masked grid $\tilde{X}^{(t)} \in ([Q] \cup \{[\mathsf{Mask}]\})^{K \times L}$ together with the evidence heatmap $S \in \mathbb{R}^{L \times Q}$ and the mask ratio γ_t, and outputs logits $\Lambda^{(t)} \in \mathbb{R}^{K \times L \times Q}$ for masked-site prediction. (For interface consistency, the implementation also accepts the parity-check matrix $H \in \mathbb{F}_Q^{P \times L}$ via an embedding interface; however, MDD does not perform explicit parity propagation and treats any parity metadata only as generic context.)

Discrete token embeddings. For a mini-batch, the masked grid is represented as $\tilde{X}^{(t)} \in ([Q] \cup \{[\mathsf{Mask}]\})^{B_{\mathrm{sz}} \times K \times L}$, where $[\mathsf{Mask}]$ is an absorbing mask token. Each site (k, ℓ) is embedded as the sum of (i) a learnable symbol embedding for $\tilde{X}_{k,\ell}^{(t)}$ and (ii) a learnable positional embedding for the slot index ℓ, yielding token embeddings $\mathbf{T}^{\mathrm{tok}} \in \mathbb{R}^{B_{\mathrm{sz}} \times K \times L \times D}$. Dropout is applied after embedding and after each residual sublayer.

Evidence (magnitude) encoder. The evidence tensor is $S \in \mathbb{R}^{B_{\mathrm{sz}} \times L \times Q}$. We standardize each slot over the alphabet dimension and project it to D dimensions via a two-layer MLP with GELU:

$$\tilde{S}_{b,\ell,:} = \frac{S_{b,\ell,:} - \mu(S_{b,\ell,:})}{\sigma(S_{b,\ell,:}) + \varepsilon}, \qquad \mathbf{T}_{b,\ell}^{\mathrm{ev}} \in \mathbb{R}^D.$$

The resulting $\mathbf{T}^{\mathrm{ev}} \in \mathbb{R}^{B_{\mathrm{sz}} \times L \times D}$ is broadcast across k and combined with $\mathbf{T}^{\mathrm{tok}}$.

Syndrome context and gating. We compute a syndrome tensor $\mathrm{syn} \in [Q]^{B_{\mathrm{sz}} \times K \times P}$ from the current grid and parity matrix (same interface as parity-aware models). Each syndrome entry is embedded, pooled across parity checks, and projected to a context vector $\mathbf{T}^{\mathrm{syn}} \in \mathbb{R}^{B_{\mathrm{sz}} \times K \times D}$. To emphasize structural cues late in denoising, the syndrome context is gated by $(1 - \gamma_t)$ and broadcast across the slot dimension. This is a generic conditioning mechanism and should not be interpreted as enforcing parity constraints; MDD does not perform Tanner-graph message passing.

Time conditioning via adaLN. The mask ratio $\gamma_t \in [0, 1]$ is embedded with a sinusoidal embedding followed by an MLP to produce $\mathbf{t}(\gamma_t) \in \mathbb{R}^{B_{\mathrm{sz}} \times D}$. We condition on time through adaptive LayerNorm (adaLN) in each Transformer block. The remaining conditioning signals are concatenated and fused:

$$\mathbf{T}^{\mathrm{fuse}} = \phi\Big(\big[\mathbf{T}^{\mathrm{tok}},\ \mathbf{T}^{\mathrm{ev}},\ \mathbf{T}^{\mathrm{syn}}\big]\Big) \in \mathbb{R}^{B_{\mathrm{sz}} \times K \times L \times D},$$

where ϕ is a fusion MLP with LayerNorm, GELU, and dropout. Each Transformer block modulates its LayerNorm statistics using learned affine parameters predicted from $\mathbf{t}(\gamma_t)$.

Backbone and output. The fused grid is flattened into a length-$K \cdot L$ token sequence and processed by N_{blk} Transformer encoder blocks with adaLN conditioning. Each block uses multi-head self-attention and a position-wise FFN with GELU, with dropout and residual connections. LayerNorm statistics are modulated by the time embedding $\mathbf{t}(\gamma_t)$. A final adaLN-modulated LayerNorm and linear projection produce logits $\Lambda^{(t)} \in \mathbb{R}^{B_{\mathrm{sz}} \times K \times L \times Q}$.

Algorithm (per-step denoiser forward). Given $(\tilde{X}^{(t)}, S, \gamma_t)$, MDD computes logits $\Lambda^{(t)}$ as:

1. Embed $\tilde{X}^{(t)}$ with symbol + slot-position embeddings to form $\mathbf{T}^{\mathrm{tok}}$.
2. Encode evidence S into $\mathbf{T}^{\mathrm{ev}}$ (slot-wise standardization $\to$ MLP with GELU $\to$ LayerNorm), then broadcast across k.
3. Encode syndrome into $\mathbf{T}^{\mathrm{syn}}$, gate by $(1 - \gamma_t)$, and broadcast across ℓ.
4. Embed time γ_t into $\mathbf{t}(\gamma_t)$ via sinusoidal embedding and MLP.
5. Concatenate $[\mathbf{T}^{\mathrm{tok}}, \mathbf{T}^{\mathrm{ev}}, \mathbf{T}^{\mathrm{syn}}]$ and fuse to obtain $\mathbf{T}^{\mathrm{fuse}}$.
6. Flatten to length-$K \cdot L$ and apply N_{blk} adaLN Transformer blocks (MHA + GELU-FFN, with dropout, residual, adaLN), each conditioned on $\mathbf{t}(\gamma_t)$.
7. Apply an adaLN-modulated final layer to produce logits $\Lambda^{(t)}$ over $[Q]$ for each site.

Implementation notes. (i) MDD is deliberately generic: it does not exploit Tanner sparsity or perform explicit parity propagation in the denoiser. (ii) All MLPs and FFNs use GELU; dropout is applied after embeddings and within each residual sublayer; LayerNorm follows standard Transformer practice. (iii) MDD is used inside an iterative refinement loop (the diffusion process), but the denoiser itself is a single forward pass per step.

K.9 CIDER: constraint-aware iterative diffusion decoding for error-correcting refinement

Overview. We use CIDER to denote the default variant used in the main paper, which consists of Module A for demixing and Module B for parity-aware propagation. We also consider an *optional* memory/stabilization Module U; when enabled, we denote the variant as **CIDER +U**. See Figure 2 for the default refinement loop (main paper) and Figure 5 for the variant with Module U (appendix).

At each refinement step t, CIDER maintains a discrete masked grid $X^{(t)} \in ([Q] \cup \{[\mathsf{Mask}]\})^{K \times L}$ and constructs a latent grid $Z^{(t)} = \mathrm{Embed}(X^{(t)}) \in \mathbb{R}^{K \times L \times D}$. The denoiser processes $Z^{(t)}$ and produces token logits $\Lambda^{(t)} \in \mathbb{R}^{K \times L \times Q}$, which are used to reveal additional entries of $X^{(t)}$. The denoiser combines (i) *slot competition* (demixing) driven by evidence $S \in \mathbb{R}^{L \times Q}$ and (ii) *parity-aware propagation* on the Tanner graph using the parity-check matrix $H \in \mathbb{F}_Q^{P \times L}$. CIDER uses standard neural components (MLPs with GELU, dropout, LayerNorm, residual connections), but applies code constraints via fixed field-consistent permutations rather than learning finite-field arithmetic.

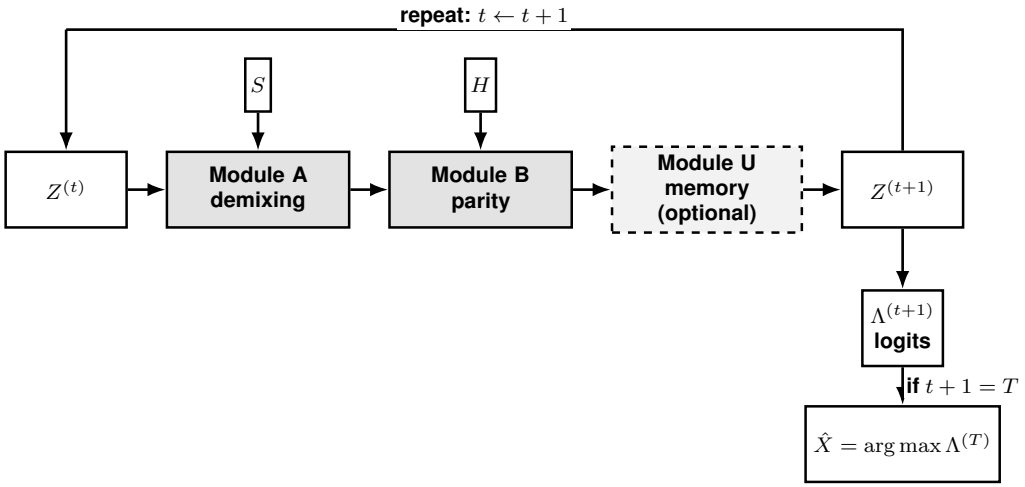


Figure 5: CIDER refinement loop with the optional memory/stabilization module (U).

Latent and symbol parameterization. We write $\hat{Z}_{k,\ell}^{(t)} \in \mathbb{R}^D$ for the refined latent vector at site (k, ℓ). Symbol logits are obtained by a shared projection

$$\Lambda_{k,\ell,a}^{(t)} = w_a^\top \hat{Z}_{k,\ell}^{(t)}, \qquad a \in [Q],$$

where $\{w_a\}_{a\in[Q]} \subset \mathbb{R}^D$ are learnable symbol vectors (equivalently, a matrix $W \in \mathbb{R}^{Q\times D}$). A learnable embedding table $E \in \mathbb{R}^{(Q+1)\times D}$ maps discrete symbols and the mask token to D dimensions when embedding a masked grid for initialization/conditioning.

Module A: slot-competition demixing. Because S is unsourced, multiple hypothesis rows can collapse onto the same high-evidence symbols. CIDER counteracts this by enforcing competition across rows. Using compatibility scores derived from the current representation $\langle Z_{k,\ell}^{(t)}, w_a\rangle + \beta_S S_{\ell,a}$, we compute per-slot responsibilities $r_{k,\ell,a}^{(t)}$ by a softmax over k (temperature τ), and form row-specific evidence summaries

$$e_{k,\ell}^{(t)} = \sum_{a\in[Q]} r_{k,\ell,a}^{(t)}\, S_{\ell,a}\, v_a,$$

where $\{v_a\}_{a\in[Q]} \subset \mathbb{R}^D$ are learnable symbol embeddings. The latent $Z_{k,\ell}^{(t)}$ and summary $e_{k,\ell}^{(t)}$ are combined using a per-site gated update: a gating network (MLP with GELU and dropout) produces a gate $\beta_{\mathrm{demix},k,\ell}^{(t)} \in [0,1]^D$ and updates

$$\tilde{Z}_{k,\ell}^{(t)} = \beta_{\mathrm{demix},k,\ell}^{(t)} \odot Z_{k,\ell}^{(t)} + \big(1 - \beta_{\mathrm{demix},k,\ell}^{(t)}\big) \odot e_{k,\ell}^{(t)}.$$

Module B: parity-aware propagation. CIDER propagates parity information along Tanner edges using the parity-check matrix $H \in \mathbb{F}_Q^{P\times L}$. For non-binary LDPC codes, each nonzero coefficient $\alpha \in \mathbb{F}_Q^\times$ induces a permutation Π_α of the Q symbols. CIDER precomputes these permutations once (using the same $\mathbb{F}_Q$ convention as H) and applies them as fixed operators. Conceptually, coefficient actions are implemented as a linear–permutation–linear map in symbol space:

$$T_\alpha(x) \triangleq W^\top \Pi_\alpha W x, \qquad x \in \mathbb{R}^D.$$

For each check node, CIDER aggregates neighbor information with an *extrinsic* pattern (excluding the edge being updated), implemented by attention over the incident edges plus a residual MLP (GELU, dropout, LayerNorm). The resulting check-to-variable messages are denormalized by the inverse coefficient action and scatter-added back to variable sites, followed by a fusion MLP to update the site latents. We denote the output of this module by $\hat{Z}^{(t)} \in \mathbb{R}^{K\times L\times D}$, which is the latent representation used for the logit projection.

Memory and stabilization (Module U). (**Optional; CIDER +U only.**) To stabilize refinement across diffusion steps, CIDER applies a GRU cell for recurrent update: a GRU cell produces an update gate $\beta^{(t)}_{\mathrm{gru},k,\ell} \in [0,1]^D$ and a proposal $\bar{Z}^{(t)}_{k,\ell}$, yielding

$$Z^{(t+1)}_{k,\ell} = \big(1 - \beta^{(t)}_{\mathrm{gru},k,\ell}\big) \odot Z^{(t)}_{k,\ell} + \beta^{(t)}_{\mathrm{gru},k,\ell} \odot \bar{Z}^{(t)}_{k,\ell}.$$

In the main text, we report the default CIDER variant without Module U and defer this ablation to Appendix Section L.2.

Algorithm (one denoiser step). For a mini-batch of size B_{sz}, let $Z^{(t)} \in \mathbb{R}^{B_{\mathrm{sz}} \times K \times L \times D}$, $S \in \mathbb{R}^{B_{\mathrm{sz}} \times L \times Q}$, and $H \in \mathbb{F}_Q^{P \times L}$. One denoiser step stacks N_{layer} structured layers, each with Module A and two Tanner-graph message-passing passes, before producing logits $\Lambda^{(t)}$ as:

1. **Intermediate logit projection.** Compute intermediate logits $\bar{\Lambda}^{(t)}$ from $Z^{(t)}$ using the shared symbol projection W. These logits are used solely for demixing.
2. **Demixing.** Compute responsibilities $r^{(t)}_{k,\ell,a}$ from $\bar{\Lambda}^{(t)}$ using a softmax over k, form evidence summaries $e^{(t)}_{k,\ell}$ from S, and blend them into $Z^{(t)}$ with a gated interpolation to obtain $\tilde{Z}^{(t)}$.
3. **Parity propagation.** Within each structured layer, perform two Tanner-graph message-passing passes using H: normalize by fixed permutations from H, aggregate extrinsically with attention and MLPs, inverse-permute, scatter-add, degree-normalize, and fuse. The final layer yields $\hat{Z}^{(t)}$.
4. **Final logit projection.** Compute final logits $\Lambda^{(t)}_{k,\ell,a} = w_a^\top \hat{Z}^{(t)}_{k,\ell}$. These final logits are used for masked-token reveal.
5. **Optional stabilization (CIDER +U only).** If Module U is enabled, apply the gated memory update to the persistent continuous state. The default CIDER variant does not carry a continuous hidden state across steps and re-embeds $Z^{(t+1)}$ from the updated discrete grid $X^{(t+1)}$.

In the default masked-diffusion loop, the decoder uses $\Lambda^{(t)}$ to update the discrete grid $X^{(t)}$ and re-embeds the next latent state from that grid. The optional CIDER +U variant additionally carries its recurrent continuous state across steps.

Implementation notes. (i) All MLP submodules use GELU and dropout, with LayerNorm and residual connections in standard practice. (ii) Coefficient handling in $\mathbb{F}_Q$ is enforced by fixed permutations derived from H, ensuring field-consistent parity messaging without learning arithmetic. (iii) CIDER differs from generic denoisers primarily through (a) row-wise competition for demixing and (b) Tanner-structured parity propagation.

K.10 Metric details

Because the decoded set is unordered, $\hat{X} \in [Q]^{K \times L}$ and $X^\star \in [Q]^{K \times L}$ are defined only up to row permutation. We choose the permutation

$$\pi^\star \triangleq \arg\min_{\pi \in \mathfrak{S}_K} \sum_{k=0}^{K-1} \sum_{\ell=0}^{L-1} \mathbf{1}\Big[\hat{X}_{k,\ell} \neq X^\star_{\pi(k),\ell}\Big],$$

where $\mathfrak{S}_K$ is the set of permutations of $[K]$. All SER and CER values in the paper are computed after this matching.

L Additional results

L.1 Numerical values underlying Figure 3

The neural baselines in Figure 3 largely overlap because most one-shot neural decoders fail in this regime, producing near-identical SER and CER values. Table 10 reports the exact numerical values across all four LDPC scales for direct comparison. CIDER is the only learned decoder that achieves substantially below-floor SER/CER.

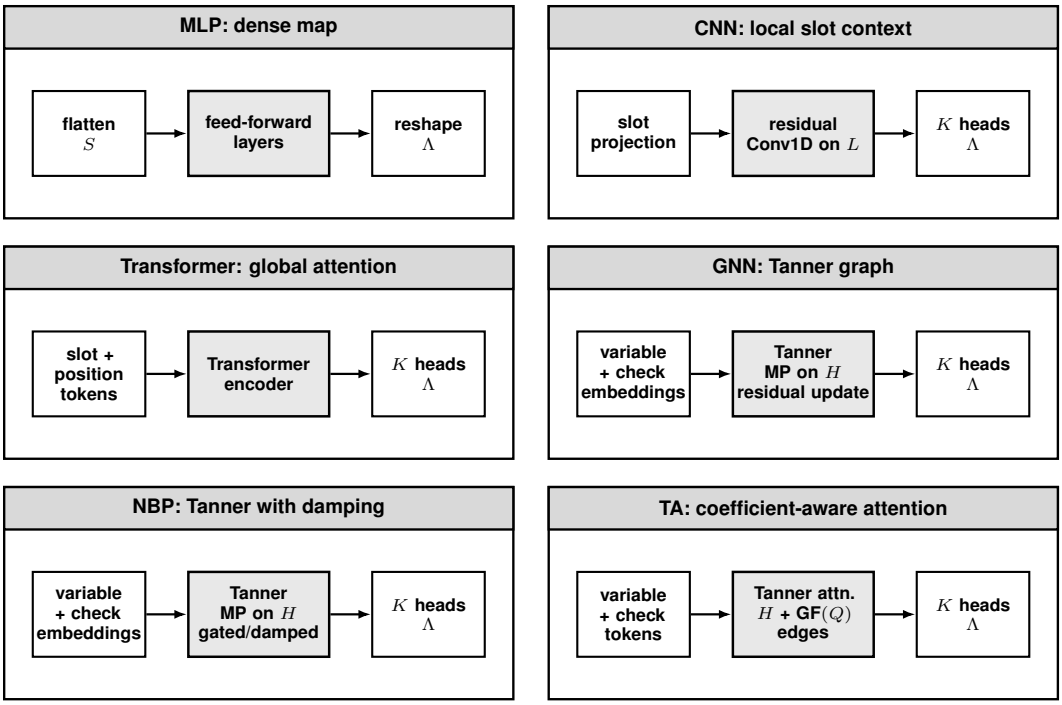


Figure 6: Compared multiuser decoder architectures. Each panel maps the shared evidence $S \in \mathbb{R}^{L\times Q}$ to $K \times L \times Q$ logits Λ. GNN, NBP, and TA additionally use the parity-check matrix H for Tanner connectivity; TA also uses its GF(Q) edge coefficients. Implementation details are given in this section.

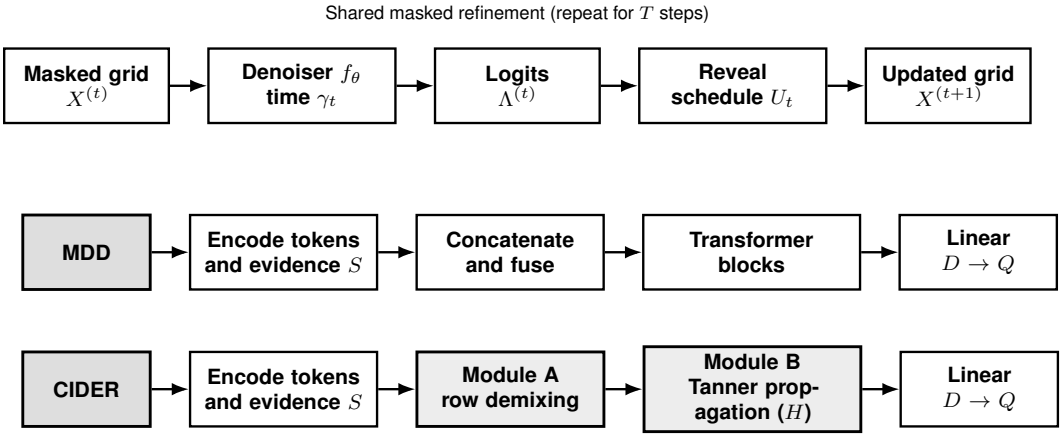


Figure 7: Shared masked-refinement loop and denoiser comparison. Both methods convert a partially masked grid and slot evidence S to logits, then reveal tokens over T steps. MDD fuses token and evidence features in a Transformer denoiser (with optional syndrome context); default CIDER adds row-wise demixing (Module A) and parity-aware Tanner propagation using H (Module B). The optional memory Module U is omitted.

L.2 Effect of the optional stabilization Module U

We also evaluate an optional memory/stabilization Module U implemented as a GRU-style update. Table 11 summarizes the effect of adding Module U across LDPC scales.

L.3 Effect of using D3PM for CIDER

This subsection reports additional results for CIDER under *uniform categorical corruption* (D3PM-style noising) [Austin et al., 2021]. Unless noted otherwise, the D3PM results use the same optional stabilization Module U (i.e., CIDER (D3PM+U)), and all other experimental settings match Section 5.

Table 10: SER/CER ($K = 2$) for CIDER and neural baselines.

Scenario (Q, L)	**Model**	**Params**	**SER (↓)**	**CER (↓)**
Tiny: $(64, 12)$	**CIDER (MDM)**	3.25M	**0.0011**	**0.0073**
	CNN	3.69M	0.4024	0.9996
	MLP	3.67M	0.6330	1.0000
	Transformer	3.63M	0.9731	1.0000
	GNN	3.54M	0.4003	0.9996
	NBP	3.81M	0.3985	0.9996
	TA	3.61M	0.4028	0.9996
	MDD	3.57M	0.4201	0.9996
Small: $(64, 18)$	**CIDER (MDM)**	4.80M	**0.0002**	**0.0013**
	CNN	5.42M	0.4186	1.0000
	MLP	5.42M	0.8942	1.0000
	Transformer	5.35M	0.9741	1.0000
	GNN	4.87M	0.4180	1.0000
	NBP	4.75M	0.4170	1.0000
	TA	5.20M	0.4210	1.0000
	MDD	5.36M	0.4375	1.0000
Moderate: $(64, 24)$	**CIDER (MDM)**	4.80M	**0.0008**	**0.0053**
	CNN	5.42M	0.4282	1.0000
	MLP	5.42M	0.9752	1.0000
	Transformer	5.35M	0.9754	1.0000
	GNN	4.87M	0.4302	1.0000
	NBP	4.75M	0.4281	1.0000
	TA	5.20M	0.4320	1.0000
	MDD	5.36M	0.4400	1.0000
Large: $(64, 48)$	**CIDER (MDM)**	6.35M	**0.0045**	**0.0270**
	CNN	7.15M	0.4470	1.0000
	MLP	7.13M	0.9774	1.0000
	Transformer	7.11M	0.9775	1.0000
	GNN	6.27M	0.4488	1.0000
	NBP	6.19M	0.4462	1.0000
	TA	7.59M	0.4513	1.0000
	MDD	7.24M	0.4601	1.0000

Table 11: Effect of adding the optional stabilization module (U) on LDPC codes. Lower is better.

Scenario (Q, L)	Model	Params	SER (↓)	CER (↓)
Tiny $(64, 12)$	CIDER (MDM; A+B)	3.25M	0.0011	0.0073
	CIDER (MDM+U; A+B+U)	3.65M	0.0014	0.0060
Small $(64, 18)$	CIDER (MDM; A+B)	4.80M	0.0002	0.0013
	CIDER (MDM+U; A+B+U)	5.40M	0.0031	0.0325
Moderate $(64, 24)$	CIDER (MDM; A+B)	4.80M	0.0008	0.0053
	CIDER (MDM+U; A+B+U)	5.40M	0.0028	0.0352
Large $(64, 48)$	CIDER (MDM; A+B)	6.35M	0.0045	0.0270
	CIDER (MDM+U; A+B+U)	7.14M	0.0102	0.0409

D3PM-style uniform-corruption baseline (D3PD). We also report D3PD, a generic diffusion baseline that mirrors MDD but replaces the absorbing-mask corruption with *uniform categorical* corruption, following the D3PM-style noising process [Austin et al., 2021]. Concretely, D3PD uses the same denoiser architecture and refinement procedure as MDD, but the forward corruption mixes each token with a uniform distribution over $[Q]$ instead of masking.

Scaling across LDPC problem sizes (Tiny/Small/Moderate/Large). Table 12 reports the same four LDPC scales used in the main paper and compares CIDER (D3PM+U) against D3PD.

Ablations (D3PM). Table 13 reports module ablations under D3PM-style (uniform categorical) corruption at $(Q, L) = (64, 12)$ with $K = 2$. Unlike the MDM setting, where the optional stabilization Module U is not always beneficial, under D3PM-style corruption the stabilization module is important for reliable refinement: removing Module U (the Module A+B variant) substantially degrades SER/CER. Table 14 additionally reports CIDER (D3PM+U) performance at two different noise levels (two SNR values), using the same architecture and training protocol. Table 15 reports an additional load check at $K = 3$ for the same setting.

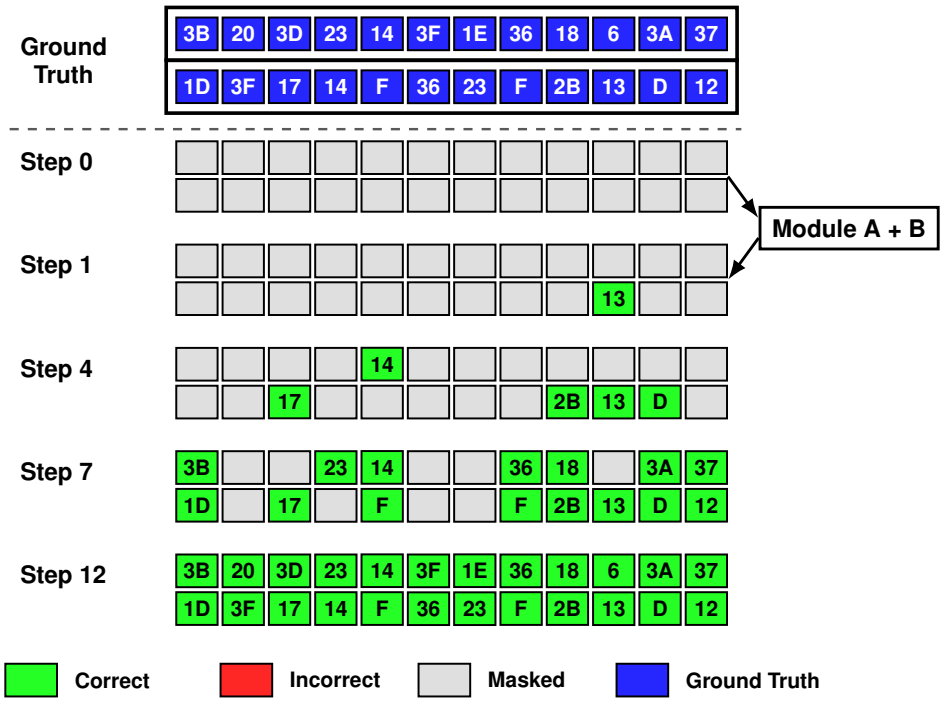


Figure 8: CIDER iterative decoding (full figure)

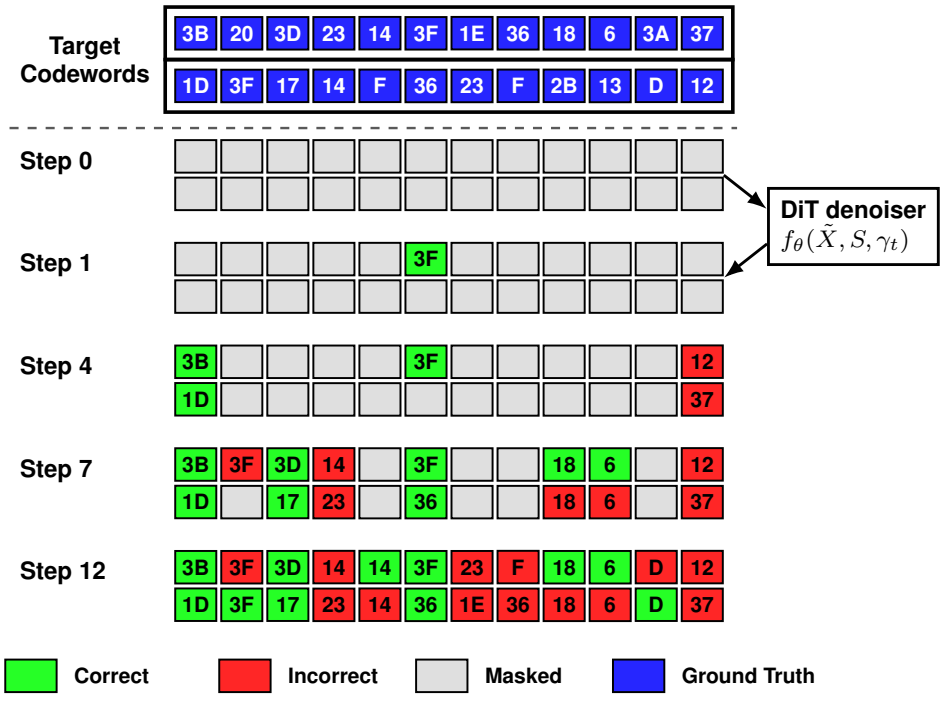


Figure 9: MDD iterative decoding

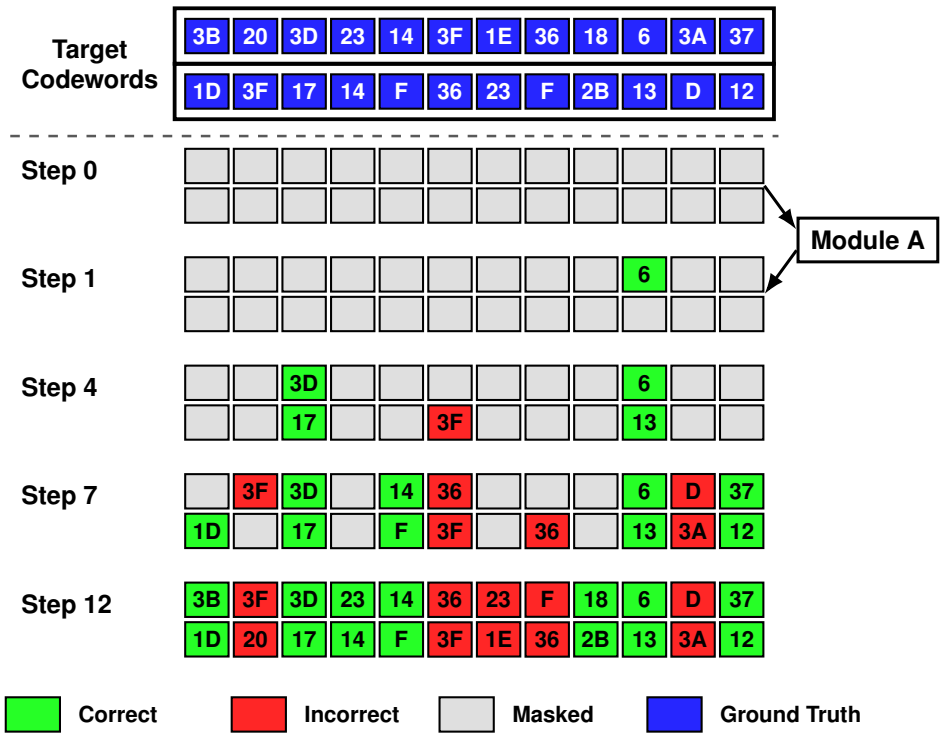


Figure 10: No parity-aware propagation (Module A only)

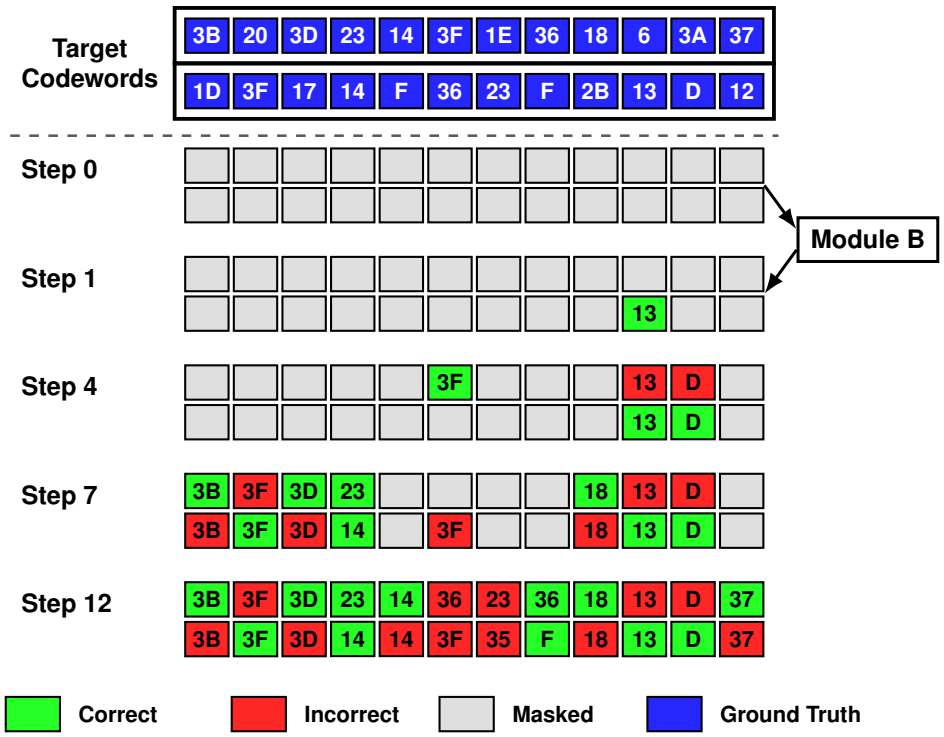


Figure 11: No demixing (Module B only)

Table 12: Scaling results for D3PM-style decoding on LDPC codes ($K = 2$). Lower is better.

Scenario (Q, L)	Model	Params	SER (↓)	CER (↓)
Tiny $(64, 12)$	CIDER (D3PM+U; A+B+U)	3.65M	0.0128	0.0339
	D3PD (uniform corruption)	3.57M	0.4721	0.9996
Small $(64, 18)$	CIDER (D3PM+U; A+B+U)	5.39M	0.0057	0.0214
	D3PD (uniform corruption)	5.36M	0.4761	1.0000
Moderate $(64, 24)$	CIDER (D3PM+U; A+B+U)	5.39M	0.0527	0.1436
	D3PD (uniform corruption)	5.36M	0.4740	1.0000
Large $(64, 48)$	CIDER (D3PM+U; A+B+U)	7.14M	0.2026	0.6192
	D3PD (uniform corruption)	7.24M	0.4853	1.0000

Table 13: Ablations of CIDER (D3PM) on LDPC codes at $(Q, L) = (64, 12)$ ($K = 2$). Lower is better.

Model	Params	SER (↓)	CER (↓)
CIDER (D3PM+U; A+B+U)	3.65M	**0.0128**	**0.0339**
No demixing (B+U)	3.74M	0.3976	0.9995
No parity-aware propagation (A+U)	1.14M	0.3867	0.9995
No memory (A+B)	3.25M	0.4294	0.9997
No diffusion (one-shot SC)	3.65M	0.4043	0.9996

L.4 Auxiliary losses for demixing and parity

We consider two auxiliary loss terms that impose global consistency through the training objective rather than through architectural constraints. Specifically, we evaluate: (i) a *demixing loss* that penalizes cosine similarity between slot predictions to discourage slot collapse and promote output diversity; and (ii) a *parity loss* that penalizes violations of the parity constraints defined by the parity-check matrix H, i.e., $Hx = 0$ over $\mathrm{GF}(Q)$. These losses are weighted by hyperparameters λ_{d} and λ_{p}, respectively, and are added to the cross-entropy objective during training. Table 16 reports results for the MDD baseline under this loss-only formulation.

L.5 Removing the first-reveal stabilization rule

We ablate the first-reveal stabilization rule defined in Appendix Section G.2. Table 17 reports performance when we disable this rule and instead run the standard confidence-based unmasking schedule directly from the all-mask initialization. Without first-reveal, SER and CER degrade markedly as K increases. This trend is more clearly seen by comparing Table 19 with Table 17: for $K = 5$, SER/CER worsen from $0.0048/0.0141$ with first-reveal to $0.1697/0.3273$ without first-reveal. These results highlight that how the early reveals are distributed across row hypotheses is critical, and increasingly so in higher-load regimes.

L.6 Scaling to larger K beyond the main setting

Table 19 extends the evaluation to larger loads at fixed $(Q, L) = (64, 12)$. Even in this higher-collision regime, CIDER continues to provide meaningful decoding accuracy. For completeness, for $K = 7$ and $K = 8$ (without PRISM) the error rates are $\mathrm{SER}/\mathrm{CER} = 0.0441/0.1006$ and $0.1339/0.2576$, respectively (Table 19).

L.7 Runtime measurement protocol

All wall-clock measurements were conducted using an NVIDIA GeForce RTX 3090 GPU (24GB) and Intel Core i5-14500 CPU. For the main $K = 2$ runtime table, inference time was averaged over 15,000 test samples for learned and BP-based methods, and over 1,000 samples for Top-J exhaustive search. For the higher-load runtime study in Appendix Section L.8, we use 15,000 samples for CIDER and 1,000 samples for FFT-BP and Top-J due to the higher runtime of the classical baselines.

Table 14: SER/CER of CIDER (D3PM+U) on LDPC codes ($K = 2$). Lower is better.

Scenario (E_b)	Model	Params	SER (↓)	CER (↓)
$\mathrm{SNR} = -2.79$ dB	CIDER (D3PM+U)	3.65M	0.0243	0.0619
$\mathrm{SNR} = -0.79$ dB	CIDER (D3PM+U)	3.65M	0.0128	0.0339

Table 15: SER/CER of CIDER (D3PM+U) on LDPC codes at $(Q, L) = (64, 12)$. Lower is better.

Scenario (K)	Model	Params	SER (↓)	CER (↓)
$K = 2$	CIDER (D3PM+U)	3.65M	0.0128	0.0339
$K = 3$	CIDER (D3PM+U)	3.65M	0.0084	0.0204

L.8 Runtime scaling to $K > 2$

To check whether the runtime advantage persists beyond the $K = 2$ setting, we repeat the wall-clock comparison at $K = 3, 4, 5$ under the same $(Q, L) = (64, 12)$ (Table 18). We measure the wall-clock time of the FFT-BP variant. The gap becomes even more pronounced: FFT-BP incurs substantially higher latency due to iterative Tanner-graph message passing, while Top-J exhaustive search becomes quickly impractical even with a small per-slot candidate list. All experiments were conducted using the same environment as above. Inference time was averaged over 15,000 test samples for CIDER, 1,000 samples for FFT-BP and Top-J Exhaustive Search (Top 3 and Top 4 for $K = 3$).

L.9 Additional sparse-graph code families: PEG-LDPC and tree codes

Progressive edge growth (PEG) constructs an LDPC Tanner graph by adding edges sequentially to increase girth and reduce short cycles. This differs in topology from the structured connectivity of the main LDPC benchmark.

The main paper focuses on non-binary LDPC codes because Module B operates on Tanner graphs. To test whether the gains depend on a particular code instance, we additionally evaluate PEG-LDPC and tree codes under the same shared evidence interface.

L.10 Failure-mode diagnostics: slot overlap and parity violation

To directly quantify the two intended failure modes of generic diffusion decoders in this setting, we measure (i) slot-overlap rate, i.e., how often different rows claim the same slot symbol evidence, and (ii) parity-violation rate, i.e., how often the decoded rows violate the code constraints. The results in Table 22 confirm that Module A primarily resolves overlap and Module B primarily resolves parity inconsistency.

L.11 Inference visualizations

Figures 8–11 visualize intermediate decoded grids during iterative inference on the Tiny setting $(Q, L) = (64, 12)$. Green/blue/red/gray indicate correct/ground-truth/incorrect/hidden tokens, respectively. Figure 8 shows that CIDER progressively resolves ambiguity and converges to two distinct, globally consistent codewords. Figure 9 shows the generic MDD baseline, which often struggles to stabilize refinement under unsourced ambiguity. Figure 11 (no demixing, i.e., removing Module A) frequently exhibits *duplicate-row collapse*, where both rows lock onto the same high-evidence symbols due to the lack of explicit row competition. Figure 10 (no parity-aware propagation, i.e., removing Module B) often yields *row swapping/mismatched assembly* across slots, where the two rows repeatedly exchange roles during refinement.

Table 16: Joint sweep over demixing and parity loss weights. Each entry reports (SER, CER) for the loss-varying MDD baseline.

$\lambda_{\mathrm{d}} \setminus \lambda_{\mathrm{p}}$	0.03	0.1	0.3	1.0
0.03	(0.7011, 1.0000)	(0.7383, 1.0000)	(0.8910, 1.0000)	(0.9505, 1.0000)
0.1	(0.6970, 1.0000)	(0.7344, 1.0000)	(0.7929, 1.0000)	(0.8762, 1.0000)
0.3	(0.6742, 1.0000)	(0.7568, 1.0000)	(0.8656, 1.0000)	(0.9612, 1.0000)
1.0	(0.7499, 1.0000)	(0.8916, 1.0000)	(0.8587, 1.0000)	(0.9445, 1.0000)

Table 17: SER/CER on LDPC codes ($(Q, L) = (64, 12)$), without first-reveal rule. Lower is better.

Scenario (K)	Model	Params	SER (↓)	CER (↓)
$K = 3$	CIDER	3.25M	0.0209	0.2162
$K = 4$	CIDER	3.25M	0.0845	0.2319
$K = 5$	CIDER	3.25M	0.1697	0.3273
$K = 6$	CIDER	4.03M	0.2778	0.4798

Table 18: Wall-clock times on LDPC codes at $(Q, L) = (64, 12)$ across per-bin loads $K \in \{3, 4, 5\}$. Time is ms/sample. Lower is better. FFT-BP runs are capped at 50 BP iterations with early exit on convergence. Top-J exhaustive search exceeded the 60-second cap for $K \geq 6$.

Method	$K = 3$			$K = 4$			$K = 5$		
	SER	CER	Time	SER	CER	Time	SER	CER	Time
CIDER	**0.0006**	**0.0044**	**5.47**	**0.0015**	**0.0058**	**10.19**	**0.0048**	**0.0141**	**15.29**
SIC-BP	0.0036	0.0063	26.46	0.0380	0.0573	90.93	0.0393	0.0738	128.76
FFT-BP	0.0036	0.0063	45.62	0.0380	0.0573	163.19	0.0393	0.0738	230.67
Top-J (Top 3)	0.1157	0.1173	6.36	–	–	–	–	–	–
Top-J (Top 4)	0.0416	0.0420	32.98	0.2190	0.2232	36.80	–	–	–
Top-J (Top 5)	0.0231	0.0233	423.02	0.0983	0.1000	427.92	0.3671	0.3736	481.71

Table 19: CIDER scales to $K = 8$, and PRISM-style remasking further improves accuracy at higher loads.

K	CIDER		**CIDER + PRISM**	
	SER (↓)	**CER (↓)**	**SER (↓)**	**CER (↓)**
2	0.0011	0.0073	–	–
3	0.0006	0.0044	–	–
4	0.0015	0.0058	–	–
5	0.0048	0.0141	–	–
6	0.0149	0.0349	**0.0064**	**0.0163**
7	0.0441	0.1006	**0.0096**	**0.0247**
8	0.1339	0.2576	**0.0166**	**0.0403**

Table 20: PEG-LDPC comparison on Tiny $(Q, L) = (64, 12)$, $K = 2$. Lower is better.

Method	**SER (↓)**	**CER (↓)**	**Time/sample (↓)**
CIDER	**0.0004**	**0.0023**	**1.39 ms**
SIC-BP	0.0015	0.0062	6.74 ms
FFT-BP	0.0015	0.0062	12.04 ms
Top-2	0.0473	0.0497	2.27 ms

Table 21: Tree-code comparison on Tiny $(Q, L) = (64, 12)$, $K = 2$. Lower is better.

Method	**SER (↓)**	**CER (↓)**	**Time/sample (↓)**
CIDER	**0.0001**	**0.0007**	**1.08 ms**
SIC-BP	0.0017	0.0065	6.82 ms
FFT-BP	0.0017	0.0065	11.56 ms
Tree-code stitching decoder	0.0474	0.0490	1.20 ms
Top-2	0.0474	0.0490	2.55 ms

Table 22: Failure-mode diagnostics on Tiny $(Q, L) = (64, 12)$, $K = 2$. Rates are percentages over the test set.

Model	**Slot-overlap rate**	**Parity-violation rate**	**SER**
MDD	63.5%	99.9%	0.4201
CIDER w/o Module A	42.6%	99.9%	0.4127
CIDER w/o Module B	2.9%	100.0%	0.3991
CIDER	**1.5%**	**0.7%**	**0.0011**

L.12 Protocol-level scalability: wrapping CIDER inside a two-step random access protocol

Motivation (scale up *without* retraining a new monolithic model). CIDER is a per-bin URA multiuser decoder: it solves *one* URA instance in *one* payload bin, and is trained for a bounded load. In the single-bin experiments of Table 19, we trained and evaluated K-specific models up to $K_{\max} = 8$ active users in that bin. Our next goal is to scale to *much larger* total frame loads while *reusing the same learned module* (i.e., the same bank of K-specific CIDER models, with PRISM for $K \geq 6$), rather than learning a brand-new architecture for large K. To do this, we wrap CIDER inside a simple two-step random access protocol that *partitions* users into multiple smaller URA subproblems.

Two-step protocol = random "binning" by a preamble. At the beginning of each frame, every active device uniformly selects one of ζ preambles (think of ζ *bins*) and transmits it in a short preamble phase. All devices that chose the same preamble are assigned to the same payload bin in the subsequent payload phase. Thus one frame induces ζ parallel or time-multiplexed payload bins, indexed by $\chi \in [\zeta]$.

Let K_{tot} denote the total number of active users in the frame (we use K_{tot} here to avoid confusion with the per-slot activity vector $U^{(\ell)}$ in Appendix Section D.3). The resulting bin loads are

$$K_\chi \triangleq \big|\{u \in [K_{\mathrm{tot}}] : \text{device } u \text{ chose preamble } \chi\}\big|, \qquad \sum_{\chi=1}^{\zeta} K_\chi = K_{\mathrm{tot}}.$$

Under uniform preamble selection, $(K_1, \ldots, K_\zeta)$ is multinomial with mean K_{tot}/ζ per bin. A convenient one-line approximation for intuition is the Poisson occupancy model:

$$K_\chi \approx \mathrm{Poisson}(\varkappa), \qquad \varkappa = K_{\mathrm{tot}}/\zeta,$$

so the *overflow probability* is $p_{\mathrm{ov}}(\varkappa) = \Pr[\mathrm{Poisson}(\varkappa) > K_{\max}]$. This overflow view organizes the two protocol-level tables. In Table 24, the bin count grows with K_{tot} ($\zeta = \lceil K_{\mathrm{tot}}/4 \rceil$), so the expected per-bin load is held near $\mathbb{E}[K_\chi] \approx 4$ and overflow is the dominant residual failure mode at every K_{tot}. The reported CER tracks the user-overflow lower bound up to a small approximately constant additive gap that reflects residual per-bin decoding error. In contrast, Table 23 fixes the bin count at $\zeta = 25$, so $\mathbb{E}[K_\chi] = K_{\mathrm{tot}}/25$ grows from 0.4 to 4 as K_{tot} sweeps from 10 to 100. At low-to-moderate K_{tot}, overflow is essentially zero (e.g., $\Pr[\mathrm{Poisson}(\varkappa) > K_{\max}] < 10^{-3}$ for $K_{\mathrm{tot}} \leq 60$) and the protocol-level CER is instead a mixture of per-load CERs $\mathrm{CER}(K_\chi)$ weighted by the bin-load distribution $K_\chi \sim \mathrm{Binomial}(K_{\mathrm{tot}}, 1/\zeta)$. Because $\mathrm{CER}(K_\chi)$ is itself mildly non-monotone in K_χ (Table 19), the protocol-level CER inherits a corresponding non-monotonicity in K_{tot}, decreasing as the bin-load distribution shifts away from low loads before the overflow penalty kicks in at high K_{tot}.

Receiver pipeline = decode each bin independently using the same learned module. For each bin χ, the receiver runs the standard two-stage pipeline: (i) a fixed symbol-wise soft detector (slot-wise AMP–MMSE) produces evidence $S^{(\chi)} \in \mathbb{R}^{L \times Q}$ for that bin, and (ii) a multiuser decoder maps $S^{(\chi)} \mapsto \hat{X}^{(\chi)} \in [Q]^{K_\chi \times L}$.

To focus on protocol partitioning and bounded-load decoding (rather than load-estimation errors), we assume the receiver knows each bin load K_χ (e.g., from preamble correlation/energy statistics). Handling errors in estimating K_χ is an orthogonal systems issue.

Crucially, we do *not* train a single monolithic model that generalizes across variable K. Instead, we deploy a bank of K-specific decoders, and in this protocol experiment we enable PRISM for the bin sizes with a trained quality head ($K_\chi \geq 6$):

$$\text{for } K_\chi \in \{1, \ldots, K_{\max}\}, \quad \hat{X}^{(\chi)} \leftarrow \big[\mathrm{CIDER}\,(+\mathrm{PRISM})\big]_{K_\chi}\Big(S^{(\chi)}\Big).$$

If a bin overflows ($K_\chi > K_{\max}$), we declare *bin overflow* and treat that bin as an erasure at the protocol level (i.e., those users are counted as failures). The full wrapper pipeline is summarized in Figure 12.

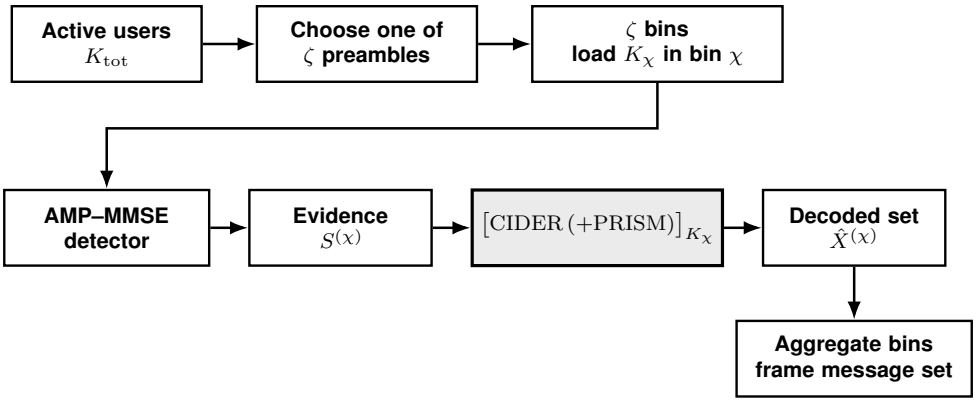


Figure 12: Two-step protocol wrapper. A preamble partitions K_{tot} active users into ζ bins. Each supported bin is decoded independently with its load-specific CIDER model (with PRISM remasking for $K_\chi \geq 6$); bins above $K_{\max}$ count as erasures.

Table 23: Stochastic binning scales to large K_{tot} by decomposing into bounded-load subproblems. Evaluated with a fixed number of bins ($\zeta = 25$); SIC-BP style decoding at comparable K_{tot} would be prohibitively slow.

K_{tot}	**SER (↓)**	**CER (↓)**
10	0.0064	0.0417
20	0.0043	0.0398
30	0.0036	0.0369
40	0.0030	0.0258
50	0.0030	0.0207
60	0.0055	0.0189
70	0.0095	0.0205
80	0.0174	0.0278
90	0.0301	0.0394
100	0.0525	0.0619

Table 24: Stochastic binning scales to many more users by decomposing into bounded-load subproblems, whereas running SIC-style decoding at comparable K_{tot} would be prohibitively slow. Evaluated for flexible number of bins $\zeta = \lceil K_{\mathrm{tot}}/4 \rceil$ so that $\mathbb{E}[K_\chi] \approx 4$.

K_{tot}	**SER (↓)**	**CER (↓)**
10	0.0027	0.0108
20	0.0225	0.0322
30	0.0219	0.0310
40	0.0423	0.0517
50	0.0389	0.0486
60	0.0493	0.0590
70	0.0423	0.0513
80	0.0502	0.0598
90	0.0467	0.0561
100	0.0525	0.0619

Experiment setup and metric (protocol-level SER/CER). We simulate K_{tot} active users per frame, assign each user to a bin by uniform preamble selection, and decode each bin using the corresponding K_χ-specific CIDER decoder when $K_\chi \leq K_{\max}$ (with additional PRISM for $K_\chi \geq 6$). Bins with $K_\chi > K_{\max}$ are treated as overflow erasures. Frame-level SER/CER are computed by aggregating errors over all K_{tot} users, counting users in overflow bins as errors. We evaluate two binning regimes: Table 23 reports results with a fixed bin count ($\zeta = 25$), and Table 24 reports results with ζ scaled with K_{tot} to maintain average per-bin load $\mathbb{E}[K_\chi] \approx 4$.

Why this matters. This wrapping experiment shows that CIDER is not only a stand-alone multiuser decoder, but also a reusable *module* inside a scalable access stack: we can support $K_{\mathrm{tot}} \gg K_{\max}$

users per frame while keeping the *worst-case per-bin* decoding cost bounded (by choosing ζ and enforcing the overflow rule).

M Additional controlled evaluations

The following experiments use the same within-bin shared-codebook decoding objective and distinguish tests that reuse a checkpoint from those trained for a target setting. They supplement the main fixed-load, calibrated-evidence experiments; the error and latency numbers are tied to their stated implementations.

M.1 Estimated and mismatched per-bin load

The Bank mode selects a K-specific model using an external energy-based load estimate $\hat{K}$. Single instead runs the $K = 8$ model for every bin, deduplicates and ranks its rows by summed slot-wise log-evidence, and retains the top $\hat{K}$ rows. Both modes use the same estimator and require no retraining. Missed-detection rate (MDR) and false-alarm rate (FAR) count errors after rectangular row matching. Single uses one checkpoint but always pays the $K = 8$ inference cost. The per-load comparison is reported in the main text (Table 4).

A deliberate one-row error in the load estimate quantifies the asymmetry: $\hat{K} = K - 1$ necessarily misses at least one of K messages, whereas $\hat{K} = K + 1$ necessarily contributes at least one false alarm among $K + 1$ reported rows. Rectangular matching counts both errors. The estimator is therefore part of the receiver rather than an oracle assumption.

Table 25: Load-mismatch sweep. Set error is (misses + false alarms)/$(K + \hat{K})$.

True K	$\hat{K} = K - 1$: MDR	$\hat{K} = K$: MDR	$\hat{K} = K + 1$: MDR/FAR/set error
3	.469	.004	.001/.250/.143
4	.291	.004	.001/.200/.112
5	.244	.017	.002/.167/.092
6	.212	.018	.002/.143/.078
7	.205	.026	.008/.125/.070

M.2 Evidence, channel, graph, and length shifts

We test the $L = 12, K = 2$ checkpoint at $E_b/N_0 = 10$ dB without weight updates, using 10,000 examples per sensing family and 2,000 per non-AMP detector. The in-distribution CER is .0073. Non-AMP outputs are converted to the same posterior statistic as the AMP evidence. The measured shifts are reported in Table 27; the uncompensated fading comparison appears in (Table 28).

Table 26: Sensitivity of CIDER to symbol-wise soft detector mismatch on Tiny $(Q, L) = (64, 12)$, $K = 2$, without retraining. I_{AMP} is the number of detector iterations.

Mismatch type	**Condition**	**SER (↓)**	**CER (↓)**
None (matched)	SNR $= -0.79$ dB, $I_{\mathrm{AMP}} = 10$	0.0011	0.0073
More AMP iters	SNR $= -0.79$ dB, $I_{\mathrm{AMP}} = 20$	0.0011	0.0075
Reduced AMP iters	SNR $= -0.79$ dB, $I_{\mathrm{AMP}} = 5$	0.0012	0.0078
SNR mismatch	SNR $= -2.79$ dB, $I_{\mathrm{AMP}} = 10$	0.0063	0.0318
SNR mismatch	SNR $= +1.21$ dB, $I_{\mathrm{AMP}} = 10$	0.0008	0.0054
SNR mismatch	SNR $= -4.79$ dB, $I_{\mathrm{AMP}} = 10$	0.1020	0.3087

These moderate shifts preserve a comparable evidence interface; they do not establish robustness to all channel models.

The graph and length transfer results are reported in the main text (Table 3). PEG-LDPC is a progressive-edge-growth Tanner graph, constructed to increase girth and suppress short cycles; the main LDPC benchmark uses structured regular connectivity designed to avoid short cycles and separately sampled nonzero field coefficients.

Table 27: Additive-noise, sensing, detector, and evidence-calibration shifts for one fixed checkpoint. Values in each CER cell correspond to the listed conditions in order.

Shift	Conditions	CER
Sensing	Complex Hadamard / QPSK Hadamard / QR Gaussian	.0113 / .0152 / .0164
Detector	Matched filter / LMMSE / whitened MAP	.0080 / .0070 / .0083
Calibration	Scale $c = .25/.5/2/4$ ($-6/-3/+3/+6$ dB)	.0119 / .0093 / .0088 / .0120
Noise	Gaussian / uniform / Student-t / impulsive	.0073 / .0077 / .0082 / .0090

Table 28: Uncompensated fading with unknown per-user gains at $L = 12, K = 2$ and $E_b/N_0 = 10$ dB. No channel estimation or fading-aware retraining is used; lower CER is better.

Decoder	AWGN	Rician $+10$ dB	Rician 0 dB	Rician -10 dB	Rayleigh
SIC-BP	.0078	.0156	.0465	.0792	.0813
CIDER	.0073	.0343	.3178	.4037	.4070

At $L = 72, 96$ the transfer latency is 17.57/34.90 ms; the length sweep starts at 3.27 ms for $L = 12$. These results support short-block portability with increasing error and compute cost. A separate capacity check on the same $L = 12$ task found CER .0011 for a larger $L = 48$ model used zero-shot, versus .0073 for the targeted $L = 12$ model. This is consistent with capacity being one possible contributor but does not isolate it. The main length sweep compares separately constructed finite graphs and scale-specific models at fixed rate $1/3$; payload grows with L and the exact-row CER criterion becomes stricter, so its non-monotonicity does not isolate a coding-gain effect.

M.3 Baseline optimization and non-diffusion controls

We batched and vectorized SIC-BP, FFT-BP, and Top-J on the same RTX 3090 used for CIDER. For this comparison, BP and Top-J timing is synchronized and uses FP32 arithmetic, while the deployed CIDER is measured in 16-bit mixed precision on the same GPU. BP folds rows into the batch, precomputes extrinsic structure, avoids per-iteration host synchronization, and exits after a valid syndrome. Top-J enumerates in parallel with memory chunking. These measurements are implementation-specific and do not imply an operation-count advantage over FFT-BP. For CIDER, the reported accuracy uses the original sampler, while optimized latency uses a vectorized sampler. The latter can break tied confidence scores differently and is therefore not bit-identical; a 512-frame Tiny check found similar CER despite differing decoded frames.

The per-scale same-GPU latency comparison is reported in the main text (Table 1).

Table 29: Amortized batch-128 time (ms/sample) and reserved GPU memory (MB), ordered SIC-BP / FFT-BP / CIDER. The memory comparison exposes the larger short-length footprint of the learned decoder.

L	Batch-128 time (ms/sample)	Reserved GPU memory (MB)
12	.72 / 1.23 / .67	32.7 / 23.2 / 58.4
18	1.79 / 3.22 / 1.57	57.8 / 48.4 / 83.7
24	14.25 / 25.77 / 3.63	90.8 / 81.7 / 97.6
48	56.07 / 101.97 / 12.26	316.6 / 307.2 / 168.7

For learned baselines, two training profiles were selected using seed-0 validation SER (CER as tie-break), without test-set selection. NBP chose cosine decay with no warmup; Tanner-Attention chose a 10-epoch warmup followed by cosine decay. Both used the same 10-iteration AMP data at $Q = 64, L = 12, K = 2$, a 70,000/15,000/15,000 train/validation/test split, learning rate 10^{-3}, batch size 128, and 100 epochs. Weight decay was 10^{-2} for no-warmup NBP and Tanner-Attention, and 10^{-4} for warmup NBP. NBP, Tanner-Attention, and parameter-matched MDD ($D = 192$, four layers and four heads) were evaluated over five seeds on the same test split with Hungarian matching and 16-bit mixed precision. Their mean CERs remain near one; this conclusion is specific to the tested architectures and training profiles. The matched-module controls in Table 2a separate masked refinement from structural iteration: one-shot A+B has SER/CER .1013/.3979, one-update iterative

Table 30: Reported operating-point latency in ms/sample for CER $\leq .05$. Floor denotes a method that did not reach the target CER; the > 60 s Top-J entries exceeded the time cap. CIDER time includes remasking where deployed.

Setting	SIC-BP	Top-J	CIDER
$K = 4, L = 12$	90.9	> 60 s ($J \geq 6$)	10.19
$K = 7, L = 12$	525	> 60 s	31.90
$K = 8, L = 12$	Floor: .26	> 60 s	63.00
$K = 2, L = 18$	14.95	1197.9	3.12
$K = 2, L = 24$	89.81	> 60 s ($J \geq 3$)	4.31
$K = 2, L = 48$	Floor: .26	> 60 s	10.08

Table 31: Five-seed neural baseline evaluation (mean $\pm$ standard deviation).

Model	SER	CER
NBP (3.81M parameters)	$.3988 \pm .0006$	$.9995 \pm .0002$
Tanner-Attention (3.61M)	$.4022 \pm .0008$	$.9995 \pm .0001$
MDD (3.57M)	$.4235 \pm .0061$	$.9995 \pm .0001$

A+B .1039/.4093, and twelve-update iterative A+B .3863/.9989, versus .0011/.0073 for CIDER. Generic masked diffusion without A+B has .4201/.9996. The existing D3PM-style no-diffusion one-shot control gives .4043/.9996, and loss-only demixing/parity regularization yields CER 1.0000. These results support the joint use of masked refinement and the two structural modules, without claiming that all possible iterative decoders have been ruled out.

M.4 Propagation depth, remasking cost, and collision failures

The implementation uses $N_{\text{layer}} = 4$ structured layers for Tiny; each layer applies two Tanner passes, giving eight passes per masked-refinement step. The following CER sweep separates this propagation depth from the number T of refinement steps. The default Tiny point is eight Tanner passes and $T = 12$. The benefit grows with load but adds latency, especially at $K = 8$. A separate $K = 2$

Table 32: CER by Tanner passes per refinement step (rows) and masked-refinement steps T (columns).

Passes $\setminus$ T	4	8	12	16	20	24
2	.5388	.0497	.0155	.0130	.0126	.0127
4	.4390	.0258	.0104	.0099	.0101	.0101
8	.3025	.0391	.0073	.0067	.0070	.0074

collision sweep uses 5,000 samples per cell and fixes the number of shared symbols between the two transmitted length-12 codewords. Table 34 reports both symbol and codeword errors. Thus CIDER leads at low overlap, but SIC-BP is stronger when the two codewords share two or three symbols.

Table 33: Quality-guided remasking at higher loads. For PRISM-on rows, time includes scoring, row selection, and triggered decoding passes; all times are ms per sample on the RTX 3090.

K	T	PRISM	SER/CER	Time
2	12	off	.0011/.0073	1.51
3	24	off	.0006/.0044	5.47
4	42	off	.0015/.0058	10.19
5	60	off	.0048/.0141	15.29
6	50	off/on	.0149/.0349 → .0064/.0163	17.40/19.79
7	62	off/on	.0441/.1006 → .0096/.0247	24.31/31.90
8	74	off/on	.1339/.2576 → .0166/.0403	30.77/63.00

Table 34: Collision-pattern failure analysis at $K = 2, L = 12$. Lower error is better.

Shared symbols	Hamming distance	CIDER SER/CER	SIC-BP SER/CER
0	12	.0005/.0030	.0040/.0155
1	11	.0026/.0248	.0067/.0252
2	10	.0199/.0823	.0124/.0543
3	9	.0593/.2870	.0060/.0347

N Limitations

CIDER is studied as a shared-codebook URA outer decoder that consumes a slot-wise evidence matrix. The main receiver uses a fixed AMP front end and synthetic sparse-graph codes under a calibrated AWGN model; the detector, sensing, calibration, and additive-noise shifts above are controlled tests rather than an end-to-end learned receiver evaluation. Unknown per-user gains without channel estimation sharply degrade CER in the fading stress test, so a fading-aware front end or retraining would be needed there. High codeword overlap also exposes a failure mode relative to SIC-BP.

The decoder must know or estimate per-bin load. The energy estimator is imperfect, and underestimating load incurs missed messages; the Single $K = 8$ checkpoint supports variable loads only with the $K = 8$ compute cost. The model uses sparse parity-check structure, and graph-specific or length-specific training still improves accuracy over zero-shot transfer. The $L = 72, 96$ transfer tests use only 256 samples each and do not establish reliable asymptotic scaling. Protocol-level operation beyond $K_{\max} = 8$ relies on stochastic binning and overflow handling.

Our latency and memory comparisons are measurements of the reported implementations on an RTX 3090, not hardware-independent complexity rankings. Quality-guided remasking improves higher-load accuracy but adds scoring, selection, and repeated decoding passes; at $K = 8$ its measured time increases from 30.77 to 63.00 ms/sample. The optimized GPU classical baselines narrow some gaps, and CIDER uses more reserved GPU memory at the shortest lengths.

AI Use Statement

Generative AI tools assisted with drafting and editing this camera-ready revision from the authors' manuscript and rebuttal. The experimental results are author-provided, and the authors take responsibility for checking the final text, data, and citations.

O Impact Statement

This paper addresses the reliable recovery of many short, uncoordinated messages from a single noisy superposition. This decoding problem arises in large-scale wireless access when coordination overhead is undesirable or infeasible. Improving recovery accuracy and latency can reduce retransmissions and access delays, which may translate into better system reliability and lower energy use in battery-constrained deployments. Beyond wireless communication, the approach frames the task as conditional discrete infilling under strong global constraints. This structure also appears in combinatorial inference problems where local evidence must be assembled into globally consistent

solutions, so the methodology may generalize to other set-structured or constraint-satisfaction settings. Responsible deployment should pair such methods with appropriate access control, auditing, and governance. Potential negative impacts include misuse in unauthorized access systems or deployment under poorly characterized channels, which could lead to unfair access failures or degraded reliability.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction accurately describe CIDER as a learned masked-diffusion outer decoder and limit the claims to the evaluated URA/joint multiuser decoding setting under a shared AMP evidence interface. The stated reliability, runtime, module-ablation, and scaling claims are supported by the corresponding experiments.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper includes a separate Limitations section in Appendix Section N.

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not present new formal theoretical results, such as theorems, lemmas, or propositions requiring proof. The mathematical content consists of problem definitions, model equations, decoding objectives, metric definitions, and computational-complexity derivations. The complexity assumptions and derivations are provided explicitly in Appendix Section I.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper provides the experimental setup, data-generation pipeline, channel model, sensing matrix construction, fixed AMP–MMSE inner detector, LDPC code scales, training and inference schedules, model sizes, evaluation metrics, row-matching procedure, baseline definitions, and runtime measurement protocol in the main text and appendices.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code is made available through the public repository link in the abstract.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The main text summarizes the experimental setting, including the fixed AMP evidence interface, evaluated LDPC scales, compared decoder families, training/inference protocol, and evaluation metrics. Full details are provided in the appendix, including data generation, train/validation/test splits, sensing matrix construction, AMP–MMSE evidence construction, code parameters, optimizer, learning rate schedule, batch size, number of epochs, EMA, precision, mask-ratio sampling, inference schedules, model sizes, and baseline architectures.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in the appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

Justification: The current version reports point estimates on held-out test sets but does not include error bars, confidence intervals, or significance tests. Although the claims are supported across multiple experimental settings, statistical uncertainty is not explicitly quantified.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper reports the computing environment and the runtime measurement protocol. Experiments were run on a single NVIDIA GeForce RTX 3090 GPU with 24GB of memory and an Intel Core i5-14500 CPU. Sufficient information about compute is described in Section J.5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more computing than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The authors have reviewed the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper includes an Impact Statement in the Appendix.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release pretrained generative models, scraped datasets, or other high-risk artifacts for misuse.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [N/A]

Justification: The paper does not use existing datasets, pretrained models, or third-party research assets as experimental inputs.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The paper introduces and releases code for the CIDER decoder and the synthetic URA/LDPC simulation pipeline through the public repository. The repository includes documentation for reproducing the main experiments, including dataset generation, model configurations, training and inference scripts, and evaluation procedures.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing, human-subject experiments, user studies, or data collected from human participants.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing, user studies, human-subject experiments, or data collected from human participants.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development and experiments do not use LLMs as an important, original, or non-standard component. Any LLM assistance, if used, was limited to writing, editing, or formatting.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.