Structured Masked Diffusion for Joint Multiuser Decoding

Anonymous Authors[1]

Abstract

In joint multiuser decoding, a receiver recovers a set of messages from a single noisy aggregate of many simultaneous transmissions. Classical decoders rely on rule-based mechanisms such as successive interference cancellation, joint belief propagation, or list recovery, all of which become brittle or expensive as ambiguity increases. We propose CIDER, a learned multiuser decoder with masked-diffusion refinement steps. CIDER uses demixing to prevent duplicate-row collapse and uses parity-aware propagation to provide soft guidance from the code constraints. In higher-load regimes, we further improve reliability via a lightweight quality-guided remasking step that selectively re-decodes low-confidence sequences. On commonly used error correcting codes, CIDER matches or improves on FFT-accelerated joint belief propagation-style decoding in symbol error rate while running more than $6\times$ to over $100\times$ faster, with the speedup widening as the blocklength grows. Code is available at `https://anonymous.4open.science/r/CIDER_2026/`

1. Introduction

Modern large-scale networked systems—spanning massive IoT deployments and dense sensor fields, large-scale distributed learning infrastructures, and autonomous robotic systems (Schwarting et al., 2018; Bonawitz et al., 2019; Kalør et al., 2025)—have renewed attention to the *joint multiuser decoding* problem: recovering multiple users' coded messages from a single noisy aggregate observation when many users transmit simultaneously over a shared channel. Classically studied in multiple-access information theory (Wang & Poor, 1999; Boutros & Caire, 2002), joint multiuser decoding became less central as deployed cellular standards adopted orthogonal scheduling, which avoids multi-user interference at the medium-access layer rather than resolving it at the receiver. With the emergence of large-scale networked services, where the device population is large while expected payloads are small, rendering per-device scheduling infeasible, joint multiuser decoding has re-emerged as a receiver strategy for next-generation wireless systems. When several users transmit at the same time, their signals add at the receiver, and decoding must be performed on this superposed observation (Polyanskiy, 2017).

A particularly challenging instance of this regime is *unsourced random access* (URA) (Polyanskiy, 2017), the canonical model for uncoordinated multiple access at scale. In this setting, multiple devices share a common codebook, transmit without prior coordination, and may send their codewords simultaneously in the same bin. The receiver observes only a noisy mixture of these transmissions and must recover the unordered set of transmitted messages, without knowing which devices were active or which user produced which codeword. This shared-codebook setting removes the per-user signature structure that classical joint multiuser decoders rely on, making the problem fundamentally one of joint set recovery rather than per-user codeword recovery. Practical receivers commonly decompose this problem by partitioning users into bins and performing joint multiuser decoding within each bin (Liva & Polyanskiy, 2024; Marshakov et al., 2019; Pradhan et al., 2022); this within-bin joint decoding step is the focus of our work.

A receiver decomposition handles this problem in 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: 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.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

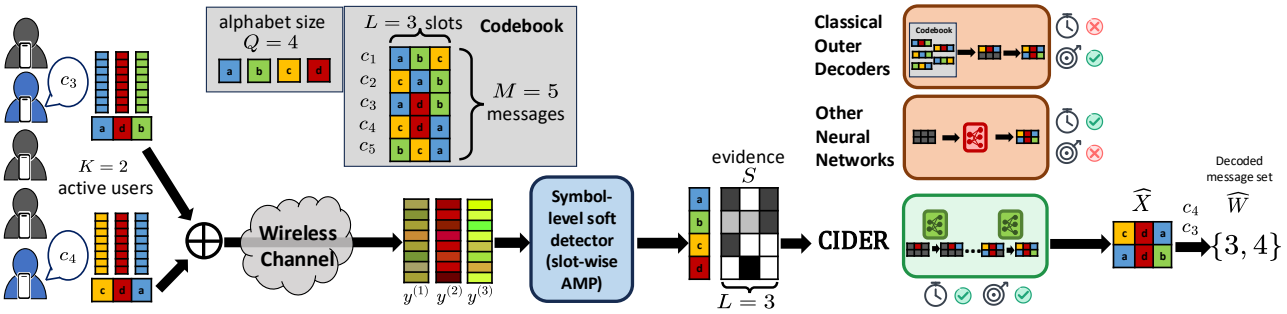


Figure 1. Two-stage receiver pipeline. Active users transmit codewords from a shared codebook over a shared channel. A fixed symbol-wise soft detector converts slot observations into the evidence matrix S, and the multiuser decoder maps S to an unordered set of decoded codewords.

Classical multiuser decoders address stability in a noisy environment 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 & Choi, 2024). Each has well-known limitations: SIC is order-dependent and propagates early errors, joint BP is computationally expensive, especially with non-binary check updates, and often requires many iterations to reach good fixed points, and stitching-based list-recovery expands combinatorially as per-slot ambiguity grows. In short-packet random access, where decoding latency is a practical constraint, these computational bottlenecks directly limit the practicality of the receiver.

So why not use a simple neural network? Because joint decoding under a shared codebook is permutation-invariant and correctness hinges on global code constraints, generic one-shot neural models fail to reliably break symmetry and produce globally valid message candidates without careful, problem-specific design (Choukroun & Wolf, 2024).

Therefore, in this work we introduce CIDER (**C**onstraint-aware **I**terative **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, improving reliability over neural and classical baselines while maintaining millisecond-scale latency.

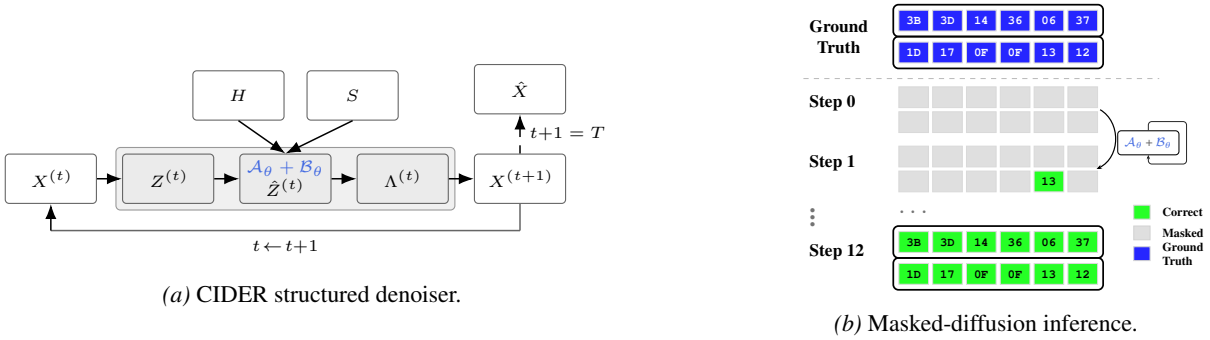
Contributions.

- We introduce masked diffusion as a learned mechanism for joint multiuser decoding. To the best of our knowledge, this is the first application of masked diffusion to wireless multiuser decoding. By incorporating domain knowledge through carefully designed masking and structural constraints, our approach addresses key limitations of *direct* diffusion-based decoders, particularly their inability to break symmetry across hypothesis rows and to enforce global code constraints.
- We instantiate the decoder for shared-codebook joint decoding and demonstrate that CIDER substantially outperforms both generic neural baselines and representative classical joint multiuser decoders, achieving significantly better accuracy while also delivering more than $6\times$ to over $100\times$ wall-clock speedup in our main LDPC setting.
- We provide extensive scaling experiments and module-wise ablations that isolate why the design is necessary (demixing, parity-aware propagation and remasking) and how performance changes with problem size and load.

2. Problem Setup

Figure 1 illustrates the complete two-stage receiver pipeline flow through a toy example. A shared codebook maps each message to a length-L codeword over a Q-ary alphabet. In each frame, K unknown active users transmit codewords through a shared channel, and the receiver must recover the unordered transmitted set. A fixed symbol-wise soft detector produces slot-wise evidence $S \in \mathbb{R}^{L\times Q}$, where $S_{\ell,a}$ scores whether symbol a appeared in slot ℓ; the multiuser decoder takes only S and outputs an unordered grid $\hat{X} \in [Q]^{K\times L}$.

The difficulty is that S is slot-local and unsourced: it identifies plausible symbols in each slot but not how to group them into globally valid messages. 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.



(a) CIDER structured denoiser.

(b) Masked-diffusion inference.

Figure 2. **CIDER overview and inference visualization.** Left: each refinement step applies the structured denoiser with demixing $\mathcal{A}_\theta$ and parity-aware propagation $\mathcal{B}_\theta$ conditioned on evidence S and parity metadata H. Right: starting from an all-[Mask] grid, CIDER progressively reveals high-confidence tokens over T masked-diffusion refinement steps.

3. 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:

$$\Lambda^{(t)} = f_\theta(X^{(t)}, S, H, t),$$
$$X^{(t+1)} = \mathrm{Reveal}(X^{(t)}, \Lambda^{(t)}).$$

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, and reveal schedule 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:

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

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

$$e^{(t)}_{k,\ell} = \sum_{a\in[Q]} r^{(t)}_{k,\ell,a}\, S_{\ell,a}\, v_a, \tag{2}$$

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

$$\tilde{Z}^{(t)}_{k,\ell} = \Phi_A\left(Z^{(t)}_{k,\ell}, e^{(t)}_{k,\ell}\right). \tag{3}$$

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 of the responsibility parameterization, gated fusion map Φ_A, and batched denoiser architecture are provided in Appendix Section K.9; additional overlap diagnostics are reported in Appendix Section L.12.

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

$$n^{(t)}_{k,j\to\ell} = \Psi_j\Big(\Big\{T_{H_{j,\ell'}}\Big(\tilde{Z}^{(t)}_{k,\ell'}\Big) : \ell' \in \mathcal{N}_{\mathrm{var}}(j)\setminus\{\ell\}\Big\}\Big), \tag{4}$$

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. This message asks: given the other slots connected to check j, what information should be sent back to slot ℓ to make the parity relation more consistent? The incoming check messages are then fused back into the token latent:

$$\hat{Z}^{(t)}_{k,\ell} = \Phi_B\left(\tilde{Z}^{(t)}_{k,\ell}, \sum_{j\in\mathcal{N}_{\mathrm{chk}}(\ell)} n^{(t)}_{k,j\to\ell}\right). \tag{5}$$

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

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

Only these final logits $\Lambda^{(t)}$ are used by the reveal rule during masked-diffusion inference. 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.

Quality-guided remasking at higher loads. At larger K, some decoded rows may remain low-confidence after the first pass. We optionally attach a lightweight quality head that scores decoded rows, remasks low-confidence rows, and re-decodes only those rows while clamping high-confidence rows. This PRISM-style (Kim et al., 2025) remasking step is used only as an inference-time reliability boost in higher-load experiments; details are given in Appendix Section G.3.

4. Experiments

This section evaluates CIDER as a learned joint multiuser decoder, with full setup and implementation details in Appendix Section H. The main experiments evaluate CIDER on shared-codebook joint decoding; a stochastic-binning scaling study is also reported.

4.1. Experimental setup

Across all experiments, 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 $K_{\mathrm{tot}} = 100$, 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.11 and L.14.

4.2. Evaluation metrics

Because the decoded message set is unordered, predicted and ground-truth grids are matched by a minimum-Hamming 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

$$\begin{aligned}\mathrm{SER}(X^\star,\hat{X}) &= \frac{1}{KL}\sum_{k,\ell}\mathbf{1}\Big[\hat{X}_{k,\ell} \neq X^\star_{\pi^\star(k),\ell}\Big],\\ \mathrm{CER}(X^\star,\hat{X}) &= \frac{1}{K}\sum_{k}\mathbf{1}\Big[\hat{X}_{k,:} \neq X^\star_{\pi^\star(k),:}\Big].\end{aligned} \tag{6}$$

Reported SER/CER are empirical averages over the test set. 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.

4.3. Compared outer decoders

We compare against three groups of outer decoders under the same evidence interface S: (i) classical decoders, including Top-J exhaustive search, SIC-BP, and FFT-BP; (ii) one-shot neural decoders, including MLP, CNN, Transformer, GNN, NBP, and Tanner-Attention; and (iii) generic masked diffusion (MDD). Full baseline definitions and hyperparameters are in Appendix Section K.

Table 1. Classical multiuser decoder comparison under the shared AMP evidence interface ($K = 2$, $Q = 64$). Time is ms/sample. SIC-BP and FFT-BP are capped at 50 BP iterations with early exit on convergence. DNF means did not finish within 24 hours.

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**	**1.26**	**0.0002**	**0.0013**	**1.83**	**0.0008**	**0.0053**	**3.20**	**0.0045**	**0.0270**	**7.66**
SIC-BP	0.0015	0.0078	96.03	0.0025	0.0081	165.76	0.0031	0.0076	655.11	0.1144	0.2680	8604.10
FFT-BP	0.0015	0.0078	8.34	0.0025	0.0081	15.19	0.0031	0.0076	59.59	0.1144	0.2680	767.51
Top-J (Top 2)	0.0476	0.0504	73.98	0.0711	0.0706	8042.65	0.0500	0.0500	77611.5	–	–	DNF
Top-J (Top 3)	0.0095	0.0093	9694.09	–	–	DNF	–	–	DNF	–	–	DNF

CIDER uses the same refinement loop as MDD but replaces the per-step denoiser with a latent denoiser tailored to joint multiuser decoding under a shared codebook that performs demixing (Module A) and parity-aware propagation (Module B).

4.4. Training and inference protocol

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.5 and K.

4.5. Main results

We organize the main results around five questions: (1) whether generic neural decoders work, (2) how CIDER compares with classical decoders, (3) how much runtime is saved, (4) which mechanisms matter, and (5) whether the method scales to many users.

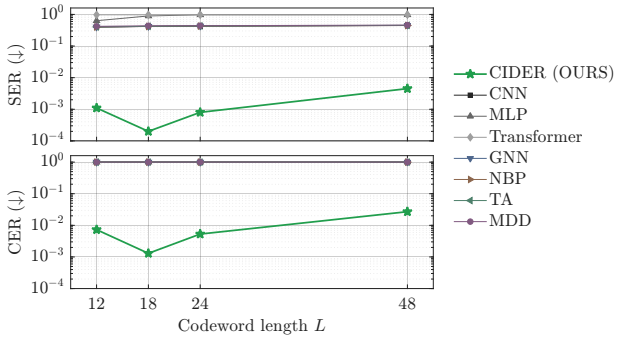


Figure 3. **CIDER vs. neural baselines across code lengths ($K = 2$). Top:** SER vs. code length L at $K = 2$; CIDER achieves low SER while other neural decoders fail. **Bottom:** CER vs. code length L at $K = 2$. Numerical values are in Appendix Section L.1.

Even at $K = 2$, other neural baselines collapse. Fig. 3 asks whether generic neural decoders can solve the smallest nontrivial multiuser setting before scaling to heavier loads. They cannot: this decoding problem is permutation-invariant, globally constrained, and driven only by slot-local unsourced evidence S, so generic one-shot networks and generic MDD collapse to overlapping high-error curves under collisions and false alarms. In contrast, CIDER remains in the low-error regime across LDPC code lengths by combining fixed-step refinement with explicit demixing and Tanner-graph parity propagation. Full numerical values are reported in Appendix Section L.1, with additional studies in Appendix Section L.

CIDER remains reliable as classical decoding becomes slow or brittle. Table 1 compares CIDER against representative classical multiuser decoders under the same shared evidence interface. Across all four code lengths, CIDER achieves the best SER/CER while remaining substantially faster. The gap widens as L grows: FFT-BP reduces the BP constant compared to SIC-BP but remains much slower than CIDER because it still relies on iterative non-binary BP inside a sequential wrapper. The full Top-J results in Appendix Section L.2 further show that Top-J search becomes intractable as the code length grows.

Table 2. Masked diffusion and structured modules are complementary on Tiny $(Q, L) = (64, 12)$, $K = 2$.

Model	Diffusion?	$\mathcal{A}$?	$\mathcal{B}$?	SER (↓)	CER (↓)
CIDER	✓	✓	✓	**0.0011**	**0.0073**
No demixing ($\mathcal{B}$ only)	✓	×	✓	0.4127	0.9993
No parity-aware prop. ($\mathcal{A}$ only)	✓	✓	×	0.3991	0.9993
MDD (generic diffusion)	✓	×	×	0.4201	0.9996
One-shot $\mathcal{A} + \mathcal{B}$	×	✓	✓	0.1013	0.3979
Iterative $\mathcal{A} + \mathcal{B}$ (L=1 model, 12 iters)	×	✓	✓	0.1039	0.4093
Iterative $\mathcal{A} + \mathcal{B}$ (L=12 model)	×	✓	✓	0.3863	0.9989

No exhaustive search, no sequential BP: millisecond decoding. All wall-clock measurements were conducted using an NVIDIA GeForce RTX 3090 GPU (24GB) and an Intel Core i5-14500 CPU. 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. Runtime is a key bottleneck for classical multiuser decoding. As shown in Table 1, CIDER achieves the best SER/CER across all four code lengths while decoding in 1.26–7.66 ms per sample. The runtime gap widens with the blocklength: at the smallest setting, CIDER is about 76× faster than SIC-BP, 7× faster than FFT-BP, and over 7,000× faster than Top-3 search; at the largest setting ($L = 48$), the gap grows to over 1,000× versus SIC-BP and 100× versus FFT-BP. The same comparison at $K > 2$ is deferred to Appendix Section L.9.

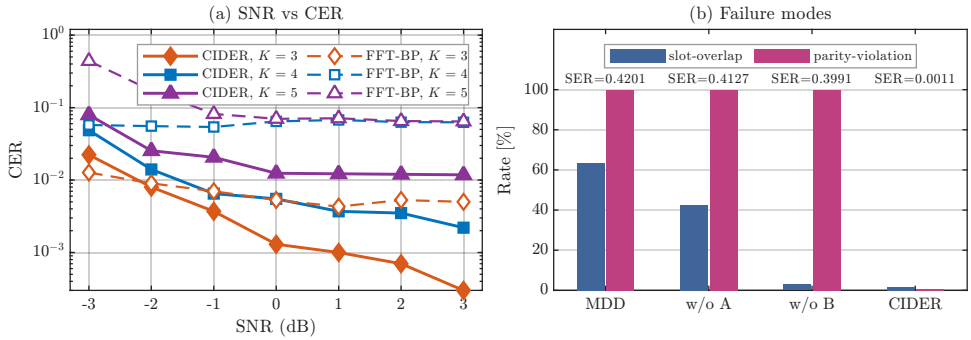


Figure 4. **Classical scaling and CIDER mechanism. Left:** CER (= PUPE) versus per-user per-channel-use SNR. CIDER is competitive at $K = 3$ and exhibits a growing advantage at higher per-bin loads ($K = 4, 5$); the model is trained at a single fixed operating point (SNR $= -0.79$ dB ($E_b/N_0 = 10$ dB)) and evaluated across the sweep without retraining. Runtime details are in Appendix Section L.9. **Right:** Failure-mode diagnostics on Tiny $(Q, L) = (64, 12)$, $K = 2$. Module $\mathcal{A}$ primarily resolves duplicate-row overlap; Module $\mathcal{B}$ restores parity consistency; only the full CIDER achieves both.

Mechanism: separate the users, then enforce the code. Fig. 4(a) additionally shows CER versus signal-to-noise ratio (SNR) for multiple per-bin loads. Fig. 4(b) visualizes the two failure modes of generic diffusion—duplicate-row overlap and parity inconsistency—and shows how the two CIDER modules address them: Module $\mathcal{A}$ separates competing user hypotheses, while Module $\mathcal{B}$ restores code consistency.

Table 2 shows that masked diffusion and problem-specific structure are complementary. Removing either Module $\mathcal{A}$ or Module $\mathcal{B}$ collapses performance, generic MDD also fails, and non-diffusion variants with the same modules do not recover CIDER's gains.

Table 3. Tree-code baselines on Tiny $(Q, L) = (64, 12)$, $K = 2$. CIDER matches LDPC speed and decoding quality on the tree-code family, outperforming code-specific stitching baselines.

Method	**SER (↓)**	**CER (↓)**	**ms (↓)**
CIDER	**0.0001**	**0.0007**	**1.08**
FFT-BP	0.0017	0.0065	7.78
Stitch	0.0474	0.0490	1.20
Top-2	0.0474	0.0490	75.2

Beyond one LDPC instance: scalable binning, robustness, and higher load. On the tree-code instance, CIDER achieves substantially lower SER/CER than the tree-code stitching decoder and Top-J search (Table 3), while retaining millisecond-scale runtime, suggesting the gains generalize beyond a single code construction.

Beyond the single-bin LDPC benchmark, Table 4a shows that CIDER remains effective as the per-bin load increases up to $K = 8$, with mild non-monotonicity at low K reflecting per-load training variance. At higher loads, PRISM-style quality-guided remasking further improves reliability by selectively re-decoding low-confidence rows while clamping high-confidence rows.

We then wrap CIDER in a stochastic-binning protocol (Marshakov et al., 2019; Vem et al., 2019). Following the bin-index notation in Appendix Section L.14, let K_χ denote the load of bin $\chi \in [\zeta]$. We scale the number of bins ζ with K_{tot} so that the average per-bin load is $\mathbb{E}[K_\chi] \approx 4$. Each bin runs an independent CIDER decoder trained for up to $K = 8$ users; bins exceeding this cap are declared erasures. Table 4b reports system-level SER/PUPE (= CER) for K_{tot} up to 100. Additional studies in Appendix Sections L.7, L.10, L.11 and L.14 report robustness to AMP/SNR mismatch, PEG-LDPC and tree-code experiments, and full protocol-level stochastic-binning details.

Table 4. **CIDER scales with the number of users.** (a) Single-bin load scaling up to $K = 8$ with PRISM-style remasking. (b) Protocol-level scaling to $K_{\text{tot}} = 100$ via stochastic binning.

(a) Load scaling (+PRISM).

K	CIDER		+PRISM	
	SER	CER	SER	CER
2	.0011	.0073	–	–
4	.0015	.0058	–	–
6	.0149	.0349	**.0064**	**.0163**
8	.1339	.2576	**.0166**	**.0403**

(b) Protocol scaling.

K_{tot}	SER	PUPE
20	0.0225	0.0322
50	0.0389	0.0486
80	0.0502	0.0598
100	0.0525	0.0619

5. Conclusion

We introduced CIDER, a learned joint multiuser decoder based on masked-diffusion refinement, instantiated for shared-codebook multiuser decoding. By combining row-wise demixing with parity-aware Tanner-graph propagation, CIDER resolves the two main failure modes of generic diffusion: duplicate-row collapse and parity inconsistency. Under a shared AMP evidence interface, CIDER improves reliability over neural and classical baselines while maintaining millisecond-scale decoding. Future work includes joint two-stage training and evaluation under richer channel models.

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.

References

Amalladinne, V. K., Chamberland, J.-F., and Narayanan, K. R. A coded compressed sensing scheme for unsourced multiple access. *IEEE Transactions on Information Theory*, 66(10):6509–6533, 2020.

Amalladinne, V. K., Pradhan, A. K., Rush, C., Chamberland, J.-F., and Narayanan, K. R. Unsourced random access with coded compressed sensing: Integrating AMP and belief propagation. *IEEE Transactions on Information Theory*, 68(4):2384–2409, 2022.

Andreev, K., Marshakov, E., and Frolov, A. 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)*, pp. 3019–3024, 2020. doi: 10.1109/ISIT44484.2020.9174247.

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

Austin, J., Johnson, D. D., Ho, J., Tarlow, D., and van den Berg, R. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

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

Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečnỳ, J., Mazzocchi, S., McMahan, H. B., Van Overveldt, T., Petrou, D., Ramage, D., and Roselander, J. Towards federated learning at scale: System design. *Proceedings of Machine Learning and Systems*, 1:374–388, 2019.

Boutros, J. and Caire, G. 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.

Chang, H., Zhang, H., Jiang, L., Liu, C., and Freeman, W. T. Maskgit: Masked generative image transformer. *arXiv preprint arXiv:2202.04200*, 2022.

Choukroun, Y. and Wolf, L. Error correction code transformer, 2022. URL https://arxiv.org/abs/2203.14966.

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

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

Ebert, J. R., Amalladinne, V. K., Rini, S., Chamberland, J.-F., and Narayanan, K. R. Coded demixing for unsourced random access. *IEEE Transactions on Signal Processing*, 70:2972–2984, 2022. doi: 10.1109/TSP.2022.3182224.

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

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

Jiang, Y., Kim, H., Asnani, H., Kannan, S., Oh, S., and Viswanath, P. Turbo autoencoder: Deep learning based channel codes for point-to-point communication channels, 2019. URL https://arxiv.org/abs/1911.03038.

Kalør, A. E., Durisi, G., Coleri, S., Parkvall, S., Yu, W., Mueller, A., and Popovski, P. Wireless 6G connectivity for massive number of devices and critical services. *Proceedings of the IEEE*, 113(9):826–848, 2025.

Ke, M., Gao, Z., Zhou, M., Zheng, D., Ng, D. W. K., and Poor, H. V. 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.

Kim, J., Kim, S., Lee, T., Pan, D. Z., Kim, H., Kakade, S., and Chen, S. Fine-tuning masked diffusion for provable self-correction. *arXiv preprint arXiv:2510.01384*, 2025.

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

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

Lou, A., Meng, C., and Ermon, S. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

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

Nachmani, E., Beery, Y., and Burshtein, D. Learning to decode linear codes using deep learning, 2016. URL https://arxiv.org/abs/1607.04793. Presented at the Allerton Conference 2016.

Nachmani, E., Marciano, E., Lugosch, L., Gross, W. J., Burshtein, D., and Beery, Y. Deep learning methods for improved decoding of linear codes, 2018. URL https://arxiv.org/abs/1706.07043. Accepted to IEEE Journal of Selected Topics in Signal Processing.

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

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

Pradhan, A. K., Amalladinne, V. K., Narayanan, K. R., and Chamberland, J.-F. Polar coding and random spreading for unsourced multiple access. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2020. doi: 10.1109/ICC40277.2020.9148687.

Pradhan, A. K., Amalladinne, V. K., Vem, A., Narayanan, K. R., and Chamberland, J.-F. Sparse IDMA: A joint graph-based coding scheme for unsourced random access. *IEEE Transactions on Communications*, 70(11):7124–7133, 2022.

Sahoo, S. S., Arriola, M., Schiff, Y., Gokaslan, A., Marroquin, E. M., Chiu, J. T., Rush, A. M., and Kuleshov, V. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2406.07524.

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

Vem, A., Narayanan, K. R., Chamberland, J.-F., and Cheng, J. 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.

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

Yun, J. and Choi, W. 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 collects all symbols used in the main text (including in equations and tables); see Table 5.

Table 5. Notation glossary.

Symbol	Meaning
Indices and sizes	
$[M]$	Message index set $\{0, 1, \dots, M-1\}$; $M \triangleq 2^B$.
$[Q]$	Alphabet/signature index set $\{0, 1, \dots, Q-1\}$.
$[K]$	Row index set $\{0, 1, \dots, K-1\}$.
$[L]$	Slot index set $\{0, 1, \dots, L-1\}$.
$[\zeta]$	Preamble-bin index set $\{1, \dots, \zeta\}$ 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	Payload length in bits per message.
K	Number of active users/messages in a frame.
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 bins / parallel payload resources in the protocol wrapper.
L	Number of slots per frame; also codeword length.
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.
Inner detector evidence	
$Y^{(\ell)} \in \mathbb{C}^{n_s}$	Slot-ℓ received vector (random variable); $y^{(\ell)}$ denotes a realized observation when needed.
$U^{(\ell)} \in \mathbb{C}^Q$	Slot-ℓ sparse activity vector (random variable)
b_q	q-th standard basis vector in $\mathbb{C}^Q$ (used to form the activity vector $U^{(\ell)}$).
$S_{\ell,a}$	Evidence 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 $w_a^\top$).
$\mathcal{A}_\theta, \mathcal{B}_\theta$	Demixing / parity propagation modules.
$r^{(t)}_{k,\ell,a}$	Responsibility (soft assignment) used in demixing.
g_ψ	Per-token quality head used for quality-guided remasking.
$\omega^{(t)}_{k,\ell}$	Per-token correctness score predicted by g_ψ.
$\bar{\omega}_k$	Row-level confidence for row k (average of $\{\omega^{(T)}_{k,\ell}\}_{\ell \in [L]}$ over slots).
$\mathcal{K}_{\mathrm{low}} \subseteq [K]$	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.
$X^{(t)}_{\mathrm{rm}} \in ([Q] \cup \{\mathsf{[Mask]}\})^{K \times L}$	Discrete masked grid during the remasking pass (rows in $\mathcal{K}_{\mathrm{low}}$ 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.
$\mathcal{N}_{\mathrm{var}}(j)$	Variable-node neighbors of check j: $\{\ell : H_{j,\ell} \neq 0\}$.
$\mathcal{N}_{\mathrm{chk}}(\ell)$	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 $\mathcal{A}$).
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 resources, and the base station then schedules a dedicated uplink 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 resources 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 system 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 resources 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–resource 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 resource, 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 resource 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 & Polyanskiy, 2021).

C.1. URA architectural families

The recent survey (Liva & 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 & 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 & 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., 2025). 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.4.

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 $[M] \triangleq \{0, 1, \dots, M-1\}$. 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 $[Q] \triangleq \{0, 1, \dots, Q-1\}$. 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 $(w_0, \dots, w_{K-1})$ 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:

$$Y^{(\ell)} \ = \ \sum_{k=0}^{K-1} a^{(\ell)}_{c_{w_k},\ell} \ + \ \epsilon^{(\ell)}, \qquad \epsilon^{(\ell)} \sim \mathcal{CN}(0, \sigma^2 I). \tag{7}$$

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

$$U^{(\ell)} \triangleq \sum_{k=0}^{K-1} b_{c_{w_k},\ell}, \tag{8}$$

where b_q is the q-th standard basis vector. Then (7) can be rewritten as

$$Y^{(\ell)} = A^{(\ell)} U^{(\ell)} + \epsilon^{(\ell)}. \tag{9}$$

To see (9) 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)

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

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

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

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

$$\pi^\star \triangleq \arg\min_{\pi \in \mathfrak{S}_K} \sum_{k=0}^{K-1} \sum_{\ell=0}^{L-1} \mathbf{1}\left[\hat{X}_{k,\ell} \neq X^\star_{\pi(k),\ell}\right]. \tag{10}$$

All symbol- and codeword-level metrics in the main text are computed after this alignment. Equivalently, (10) 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 & 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 (8), 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*

$$\eta_{\mathrm{MMSE}}(r;\nu) \;\triangleq\; \mathbb{E}[U \mid R = r]\,. \tag{11}$$

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,

$$\eta'_{\mathrm{MMSE}}(r;\nu) \;\triangleq\; \frac{\partial}{\partial r}\,\eta_{\mathrm{MMSE}}(r;\nu), \tag{12}$$

and in the complex setting we use the standard AMP analogue of this scalar sensitivity (see Bayati & 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 (11) admit closed forms:

$$\hat{p}(r) = \frac{\rho_{\mathrm{BG}}\,\phi(r;0,\nu+\sigma_u^2)}{(1-\rho_{\mathrm{BG}})\,\phi(r;0,\nu) \;+\; \rho_{\mathrm{BG}}\,\phi(r;0,\nu+\sigma_u^2)}, \tag{13}$$

$$\eta_{\mathrm{MMSE}}(r;\nu) = \hat{p}(r)\cdot\frac{\sigma_u^2}{\nu+\sigma_u^2}\,r, \tag{14}$$

where $\phi(\cdot;0,\cdot)$ denotes the circular complex Gaussian density. For completeness, the activity posterior in (13) 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 (14) 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 (14) coordinate-wise, and uses the corresponding sensitivity term (12) (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:

$$R^{(i)} = \hat{U}^{(i)} + (A^{(\ell)})^H \tilde{Y}^{(i)}, \tag{15}$$

$$\hat{U}_a^{(i+1)} = \eta_{\mathrm{MMSE}}\Big(R_a^{(i)};\, \nu^{(i)}\Big) \qquad (a \in [Q]), \tag{16}$$

$$\tilde{Y}^{(i+1)} = Y^{(\ell)} - A^{(\ell)}\hat{U}^{(i+1)} \;+\; \frac{\tilde{Y}^{(i)}}{n_s}\sum_{a\in[Q]}\eta'_{\mathrm{MMSE}}\Big(R_a^{(i)};\, \nu^{(i)}\Big), \tag{17}$$

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 (17) 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 & 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 (13). 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

$$\mathcal{C}_{\mathrm{LDPC}} \triangleq \{\, \mathbf{c} \in \mathbb{F}_Q^L : H\mathbf{c} = \mathbf{0} \text{ in } \mathbb{F}_Q \,\}. \tag{18}$$

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

$$\mathcal{C} = \{\mathbf{c}_m\}_{m\in[M]} = \mathcal{C}_{\mathrm{LDPC}}, \qquad \text{so each } \mathbf{c}_m \text{ satisfies } H\mathbf{c}_m = \mathbf{0}. \tag{19}$$

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

$$\sum_{\ell=0}^{L-1} H_{j,\ell}\,(\mathbf{c}_m)_\ell \;=\; 0 \qquad \text{in } \mathbb{F}_Q, \tag{20}$$

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 (20) can be written more explicitly as

$$\sum_{\ell' \in \mathcal{N}_{\mathrm{var}}(j)} H_{j,\ell'}\,(\mathbf{c}_m)_{\ell'} \;=\; 0 \qquad \text{in } \mathbb{F}_Q, \tag{21}$$

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

$$\lambda_\ell(a) \;\propto\; \exp(S_{\ell,a}), \qquad a \in [Q],\ \ell \in [L], \tag{22}$$

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 (18).

Standard LDPC decoding principle: belief propagation on the Tanner graph. A classical way to enforce (18) given soft beliefs (22) 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:

$$m_{\ell\to j}(a) \;\propto\; \lambda_\ell(a) \prod_{j' \in \mathcal{N}_{\mathrm{chk}}(\ell)\setminus\{j\}} m_{j'\to\ell}(a). \tag{23}$$

The check update enforces the parity constraint by summing over assignments of neighboring variables that satisfy (21):

$$m_{j\to\ell}(a) \;\propto\; \sum_{\{a_{\ell'}\}_{\ell'\in\mathcal{N}_{\mathrm{var}}(j)\setminus\{\ell\}}} \mathbf{1}\left[H_{j,\ell}\,a \;+\; \sum_{\ell'\in\mathcal{N}_{\mathrm{var}}(j)\setminus\{\ell\}} H_{j,\ell'}\,a_{\ell'} = 0\right] \prod_{\ell'\in\mathcal{N}_{\mathrm{var}}(j)\setminus\{\ell\}} m_{\ell'\to j}(a_{\ell'}). \tag{24}$$

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

$$b_\ell(a) \;\propto\; \lambda_\ell(a) \prod_{j\in\mathcal{N}_{\mathrm{chk}}(\ell)} m_{j\to\ell}(a), \qquad a\in[Q], \tag{25}$$

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. Here and below, the symbol arguments a and $\{a_{\ell'}\}$ are interpreted as elements of $\mathbb{F}_Q$ via the fixed labeling ϕ.

How CIDER uses H. CIDER leverages the same parity structure but integrates it into the learned denoiser (Module $\mathcal{B}$) 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 first sample a sparse Tanner-graph connectivity pattern (i.e., the locations of nonzeros in H), 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 $t \in \{0, 1, \ldots, T\}$, 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:

$$p_\theta\left(\tilde{x}_j^{(s)} \mid \tilde{x}_j^{(t)}, \xi\right) \triangleq q\left(\tilde{x}_j^{(s)} \mid \tilde{x}_j^{(t)}, \hat{x}_{\theta,j}(\tilde{X}^{(t)}, \xi, t)\right). \tag{26}$$

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 $a \in [Q]$,

$$q_{\text{unif}}\Big(\bar{X}_j^{(t)} = a \,\Big|\, X_j = x_j\Big) = (1-\bar{\gamma}_t)\,\mathbf{1}[a = x_j] \;+\; \bar{\gamma}_t \cdot \frac{1}{Q}. \tag{27}$$

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:

$$\rho(t) \triangleq \frac{1}{2}\left(1 - \cos\left(\pi \frac{t}{T}\right)\right), \qquad t = 0, 1, \ldots, T. \tag{28}$$

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

$$\gamma_t^{\text{infer}} = 1 - \rho(t),$$

so the denoiser conditioning is consistent with the fraction of tokens that remain masked at each refinement step.

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 $\rho(t)$.

We choose the cosine schedule in (28) for its simple closed form and empirical stability.

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., 2025). 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 $t \in \{1,\ldots,T\}$. 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

$$p_{k,\ell,:}^{(t)} \triangleq \text{softmax}\Big(\Lambda_{k,\ell,:}^{(t)} / \tau_t\Big) \in \Delta^Q$$

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

$$\omega_{k,\ell}^{(t)} \;=\; \sigma\Big(g_\psi\Big(\hat{Z}_{k,\ell}^{(t)}\Big)\Big) \in [0,1], \tag{29}$$

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 $p_{k,\ell,:}^{(t)}$ 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 $\mathcal{S} \subseteq \{(k,\ell) : \tilde{X}_{k,\ell}^{(t)} = [\mathsf{Mask}]\}$, we sample a token prediction

$$\hat{x}_{k,\ell}^{(t)} \sim \mathrm{Cat}\Big(p_{k,\ell,:}^{(t)}\Big), \qquad (k,\ell) \in \mathcal{S}. \tag{30}$$

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

$$b_{k,\ell}^{(t)} \;\triangleq\; \mathbf{1}\Big[\hat{x}_{k,\ell}^{(t)} = X^\star_{\pi^\star(k),\ell}\Big], \tag{31}$$

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

$$\mathcal{L}_{\mathrm{qual}}(\psi) \;=\; \sum_{(k,\ell)\in\mathcal{S}} \Big(- b_{k,\ell}^{(t)} \log \omega_{k,\ell}^{(t)} - (1 - b_{k,\ell}^{(t)}) \log(1 - \omega_{k,\ell}^{(t)}) \Big). \tag{32}$$

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:

$$\bar{\omega}_k \;\triangleq\; \frac{1}{L} \sum_{\ell=0}^{L-1} \omega_{k,\ell}^{(T)}. \tag{33}$$

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:

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

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

$$X_{\mathrm{rm}}^{(0)}{}_{k,\ell} = \begin{cases} [\mathsf{Mask}], & k \in \mathcal{K}_{\mathrm{low}}, \\ \hat{X}_{k,\ell}, & k \notin \mathcal{K}_{\mathrm{low}}. \end{cases} \tag{34}$$

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

$$T_{\mathrm{rm}} = \left\lfloor T \cdot \frac{|\mathcal{K}_{\mathrm{low}}|}{K} \right\rfloor,$$

where T is the number of steps used in the initial decoding pass. This process repeats (increasing s) until $\mathcal{K}_{\mathrm{low}} = \emptyset$ 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 $\omega_{k,\ell}^{(T)} = \sigma(g_\psi(\hat{Z}_{k,\ell}^{(T)}))$ and sequence confidences $\bar{\omega}_k = \frac{1}{L}\sum_\ell \omega_{k,\ell}^{(T)}$.
3. Select $\mathcal{K}_{\text{low}}$ as the sequences with $\bar{\omega}_k < \varphi_s$.
4. Initialize $X_{\text{rm}}^{(0)}$ by remasking sequences (rows) in $\mathcal{K}_{\text{low}}$ (Eq. (34)).
5. Rerun masked-diffusion decoding from $X_{\text{rm}}^{(0)}$ 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 $\varrho = 3$ stages for $K = \{6, 7, 8\}$ in our reported results.

H. Experimental setup and implementation details

This appendix collects implementation details omitted from Section 4: (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 (8) and generate

$$Y^{(\ell)} = A^{(\ell)}U^{(\ell)} + \epsilon^{(\ell)}, \qquad \epsilon^{(\ell)} \sim \mathcal{CN}(0, \sigma^2 I). \tag{35}$$

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).

Scale-dependent dataset sizes and sensing dimensions. We use the following $(n_s, \#\text{frames})$ per scale in our implementation:

- Tiny: $n_s = 24$, 100,000 frames.
- Small: $n_s = 24$, 100,000 frames.
- Moderate: $n_s = 24$, 100,000 frames.
- Large: $n_s = 24$, 100,000 frames.

We use a 70K/15K/15K 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 DFT matrix and select the first n_s rows:

$$\mathbf{A}_{\text{DFT}} = \frac{1}{\sqrt{n_s}}\mathbf{F}_{0:n_s-1,:} \in \mathbb{C}^{n_s\times Q}. \tag{36}$$

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

H.3. 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.4. 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.5. 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 $\Omega_t \subseteq [K]\times[L]$ be the masked grid sites and let $p^{(t)}_{k,\ell,:}$ be the predicted categorical distribution at site (k,ℓ). We form the row-matching cost

$$D_{k,k'} = \sum_{\ell:(k,\ell)\in\Omega_t} \mathrm{CE}\Big(X^\star_{k',\ell}, p^{(t)}_{k,\ell,:}\Big),$$

compute

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

and minimize the matched masked cross-entropy

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

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

MDD mask-ratio sampling (training). We sample a discrete timestep t uniformly from $\{0,1,\dots,T_{\text{train}}\}$ with $T_{\text{train}} = 16$ and set

$$\gamma_t \triangleq \gamma_{\max} - (\gamma_{\max}-\gamma_{\min})\frac{t}{T_{\text{train}}}, \qquad (\gamma_{\min},\gamma_{\max}) = (0.1, 1.0). \tag{37}$$

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 $1/(T_{\text{train}}+1) \approx 5.9\%$ 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 (28).

Inference temperature annealing. We use cosine annealing for the inference temperature from $\tau_{\max}$ at the first refinement step to $\tau_{\min}$ at the final refinement step:

$$\tau_{\text{infer},t} \triangleq \tau_{\max}\, w(t) + \tau_{\min}\,\big(1-w(t)\big), \qquad w(t) \triangleq \frac{1}{2}\left(1+\cos\left(\pi\frac{t-1}{T-1}\right)\right). \tag{38}$$

The denoiser probabilities are computed as $p^{(t)}_{k,\ell,:} = \mathrm{softmax}(\Lambda^{(t)}_{k,\ell,:}/\tau_{\text{infer},t})$.

Final refinement pass. After all tokens are revealed, we run one additional forward pass at $\gamma = 0$ and output by per-site argmax.

H.6. 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 yields a $\sim 10\times$ per-edge speedup, consistent with the wall-clock speedup we observe in practice.

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 $L_{\text{info}} \triangleq L - P$.

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. Training hyperparameters

Table 8. 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 $t \sim \text{Unif}\{0, \ldots, T_{\text{train}}\}$, $\gamma_t \in [0.1, 1.0]$ after warmup
Mask warmup	4 epochs (exclude $\gamma = 1$)
$\Pr[\gamma = 1.0]$	$1/(T_{\text{train}} + 1) \approx 0.059$ after warmup
AMP iterations I_{AMP}	20

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 $\mathbb{R}^{B_{\text{sz}}\times L\times Q}$) and output token logits $\Lambda \in \mathbb{R}^{K\times L\times Q}$ (or batched as $\mathbb{R}^{B_{\text{sz}}\times K\times L\times Q}$). 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 $x = \text{vec}(S) \in \mathbb{R}^{B_{\text{sz}}\times(LQ)}$ 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 $\Lambda \in \mathbb{R}^{B_{\text{sz}}\times K\times L\times Q}$.

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:

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

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 $\Lambda \in \mathbb{R}^{B_{\text{sz}}\times K\times L\times Q}$.

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 $Q \to D_{\text{tr}}$, producing $x \in \mathbb{R}^{B_{\text{sz}}\times L\times D_{\text{tr}}}$.

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($D_{\text{tr}} \to D_{\text{tr}}$)→GELU→Dropout→Linear($D_{\text{tr}}\to Q$), and stacks the results to obtain $\Lambda \in \mathbb{R}^{B_{\text{sz}}\times K\times L\times Q}$.

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 $x \in \mathbb{R}^{B_{\text{sz}} \times L \times D_{\text{gnn}}}$. Learnable slot embeddings $s_k \in \mathbb{R}^{D}_{\text{gnn}}$ 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($D_{\text{gnn}} \to D_{\text{gnn}}$)→GELU→Dropout→Linear($D_{\text{gnn}} \to Q$), producing $\Lambda \in \mathbb{R}^{B_{\text{sz}} \times K \times L \times Q}$.

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 slot 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. Unlike Transformer-H (which uses dense check tokens and global cross-attention), 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 $\mathcal{N}_{\mathrm{var}}(j) = \{\ell \in [L] \ : \ H_{j,\ell} \neq 0\}$ denote the variable neighbors of check j, and $\mathcal{N}_{\mathrm{chk}}(\ell) = \{j \in [P] \ : \ H_{j,\ell} \neq 0\}$ 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 $\mathbf{T}^{\mathrm{chk}}$ and keys/values are $\mathbf{T}^{\mathrm{var}}$ restricted to $\mathcal{N}_{\mathrm{var}}(j)$. 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 $\mathcal{N}_{\mathrm{chk}}(\ell)$, 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. (iii) The primary difference from Transformer-H is that TA exploits Tanner sparsity explicitly via neighbor-restricted attention, rather than using dense check tokens with a global cross-attention block.

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

Figure 7 illustrates this baseline.

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 [Mask] is an absorbing mask token. Each site (k, ℓ) is embedded as the sum of (i) a learnable symbol embedding for $\tilde{X}^{(T)}_{k,\ell}$ 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:

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

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.

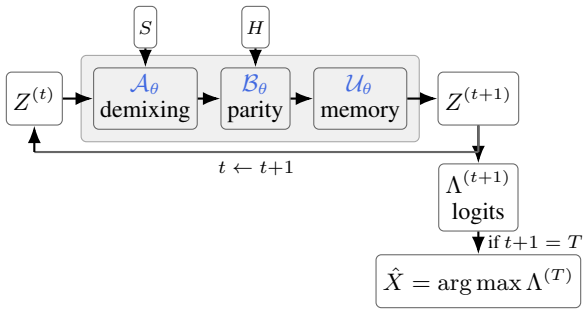


Figure 5. CIDER refinement loop with the optional memory/stabilization module (U).

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 demixing ($\mathcal{A}_\theta$) and parity-aware propagation ($\mathcal{B}_\theta$). We also consider an *optional* memory/stabilization module ($\mathcal{U}_\theta$); 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 $\mathcal{U}_\theta$ (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.

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.

Slot-competition demixing (module $\mathcal{A}_\theta$). 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 $\langle Z_{k,\ell}^{(t)}, w_a\rangle + \beta, S_{\ell,a}$, 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

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

Parity-aware propagation (module $\mathcal{B}_\theta$). 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 ($\mathcal{U}_\theta$). **(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 $\mathcal{U}_\theta$ and defer this ablation to Appendix Section L.3.

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 computes $Z^{(t+1)}$ and 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.** Perform one round of Tanner-graph message passing using H: normalize by fixed permutations from H, aggregate extrinsically with attention and MLPs, denormalize, scatter-add, and fuse to obtain $\hat{Z}^{(t)}$.
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 $\mathcal{U}_\theta$ 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 masked-diffusion loop, the decoder uses $\Lambda^{(t)}$ to select updates to the discrete grid $X^{(t)}$ and proceeds to the next refinement step, where $Z^{(t+1)}$ is re-embedded from the updated discrete state.

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 9 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.

Table 9. 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

L.2. Full classical multiuser decoder results with Top-J search

Table 10 reports the full classical comparison including Top-J exhaustive search. Top-J search is highly sensitive to both the candidate-list size J and the code length L: increasing J improves accuracy on Tiny, but the runtime grows rapidly, and the method does not finish within 24 hours for longer code lengths. This supports the main-text claim that Top-J enumeration becomes intractable as the code length grows.

L.3. Effect of the optional stabilization module $\mathcal{U}_\theta$

We also evaluate an optional memory/stabilization module $\mathcal{U}_\theta$ implemented as a GRU-style update. Table 11 summarizes the effect of adding $\mathcal{U}_\theta$ across LDPC scales.

L.4. 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 $\mathcal{U}_\theta$ (i.e., CIDER (D3PM+U)), and all other experimental settings match Section 4.

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 $\mathcal{U}_\theta$ is not always beneficial, under D3PM-style corruption the stabilization module is important for reliable refinement: removing $\mathcal{U}_\theta$ (the $\mathcal{A} + \mathcal{B}$ variant)

Table 10. Full classical multiuser decoder comparison across LDPC scales ($K = 2$, $Q = 64$). Lower is better. DNF = did not finish within 24 hours.

Scale (Q, L)	**Method**	**SER (↓)**	**CER (↓)**	**Time/sample (↓)**
Tiny $(64, 12)$	**CIDER**	**0.0011**	**0.0073**	**1.26 ms**
	SIC-BP	0.0015	0.0078	96.03 ms
	FFT-BP	0.0015	0.0078	8.34 ms
	Top-J Exhaustive Search (Top 2)	0.0476	0.0504	73.98 ms
	Top-J Exhaustive Search (Top 3)	0.0095	0.0093	9694.09 ms
Small $(64, 18)$	**CIDER**	**0.0002**	**0.0013**	**1.83 ms**
	SIC-BP	0.0025	0.0081	165.76 ms
	FFT-BP	0.0025	0.0081	15.19 ms
	Top-J Exhaustive Search (Top 2)	0.0711	0.0706	8042.65 ms
	Top-J Exhaustive Search (Top 3)	–	–	DNF ($>$ 24 h)
Moderate $(64, 24)$	**CIDER**	**0.0008**	**0.0053**	**3.20 ms**
	SIC-BP	0.0031	0.0076	655.11 ms
	FFT-BP	0.0031	0.0076	59.59 ms
	Top-J Exhaustive Search (Top 2)	0.0500	0.0500	77611.5 ms
	Top-J Exhaustive Search (Top 3)	–	–	DNF ($>$ 24 h)
Large $(64, 48)$	**CIDER**	**0.0045**	**0.0270**	**7.66 ms**
	SIC-BP	0.1144	0.2680	8604.10 ms
	FFT-BP	0.1144	0.2680	767.51 ms
	Top-J Exhaustive Search (Top 2)	–	–	DNF ($>$ 24 h)
	Top-J Exhaustive Search (Top 3)	–	–	DNF ($>$ 24 h)

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; $\mathcal{A} + \mathcal{B}$)	3.25M	0.0011	0.0073
	CIDER (MDM+U; $\mathcal{A} + \mathcal{B} + \mathcal{U}$)	3.65M	0.0014	0.0060
Small $(64, 18)$	CIDER (MDM; $\mathcal{A} + \mathcal{B}$)	4.80M	0.0012	0.0064
	CIDER (MDM+U; $\mathcal{A} + \mathcal{B} + \mathcal{U}$)	5.40M	0.0031	0.0325
Moderate $(64, 24)$	CIDER (MDM; $\mathcal{A} + \mathcal{B}$)	4.80M	0.0008	0.0053
	CIDER (MDM+U; $\mathcal{A} + \mathcal{B} + \mathcal{U}$)	5.40M	0.0028	0.0352
Large $(64, 48)$	CIDER (MDM; $\mathcal{A} + \mathcal{B}$)	6.35M	0.0045	0.0270
	CIDER (MDM+U; $\mathcal{A} + \mathcal{B} + \mathcal{U}$)	7.14M	0.0102	0.0409

substantially degrades SER/CER. Table 14 additionally reports CIDER (D3PM+U) performance at two different noise levels (two E_b values), using the same architecture and training protocol. Table 15 reports an additional load check at $K = 3$ for the same setting.

L.5. 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 incidence 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.6. Removing the first-reveal stabilization rule

The first-reveal stabilization rule commits each slot's initial symbol in isolation—no other token is unmasked in the same step—so that every slot is anchored at its most confident position before joint unmasking proceeds. Table 17 reports performance when we disable the first-reveal stabilization rule and instead run the standard confidence-based unmasking

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; $\mathcal{A}+\mathcal{B}+\mathcal{U}$)	3.65M	0.0128	0.0339
	D3PD (uniform corruption)	3.57M	0.4721	0.9996
Small $(64, 18)$	CIDER (D3PM+U; $\mathcal{A}+\mathcal{B}+\mathcal{U}$)	5.39M	0.0057	0.0214
	D3PD (uniform corruption)	5.36M	0.4761	1.0000
Moderate $(64, 24)$	CIDER (D3PM+U; $\mathcal{A}+\mathcal{B}+\mathcal{U}$)	5.39M	0.0527	0.1436
	D3PD (uniform corruption)	5.36M	0.4740	1.0000
Large $(64, 48)$	CIDER (D3PM+U; $\mathcal{A}+\mathcal{B}+\mathcal{U}$)	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; $\mathcal{A}+\mathcal{B}+\mathcal{U}$)	3.65M	**0.0128**	**0.0339**
No demixing ($\mathcal{B}+\mathcal{U}$)	3.74M	0.3976	0.9995
No parity-aware propagation ($\mathcal{A}+\mathcal{U}$)	1.14M	0.3867	0.9995
No memory ($\mathcal{A}+\mathcal{B}$)	3.25M	0.4294	0.9997
No diffusion (one-shot SC)	3.65M	0.4043	0.9996

schedule 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 $0.0101/0.0281$ (with first-reveal; Table 19) to $0.1697/0.3273$ (without first-reveal; Table 17). These results highlight that the early refinement phase is critical, and that stabilization becomes increasingly important in higher-load regimes.

L.7. 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.8. Runtime measurement protocol

All wall-clock measurements were conducted using an NVIDIA GeForce RTX 3090 GPU (24GB) and Intel Core i5-14500 CPU. 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.

L.9. 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 check the wall-clock time of 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. In particular, for $K = 3$, increasing the candidate list from Top 3 to Top 4 improves accuracy (SER/CER $0.1889/0.2850 \to 0.0812/0.1013$) but increases runtime from 9.47 s to 316.93 s (about $33\times$ slower). Even with this extra cost, Top-J Exhaustive Search (Top 4) remains far less accurate than CIDER while being orders of magnitude slower. 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.10. Robustness to slot-wise soft detector 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. The results in Table 20 show that moderate degradation

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

Scenario (E_b)	Model	Params	SER (↓)	CER (↓)
$E_b = 8$	CIDER (D3PM+U)	3.65M	0.0243	0.0619
$E_b = 10$	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

in the evidence is handled gracefully, whereas severe SNR mismatch eventually degrades performance substantially.

L.11. Additional sparse-graph code families: PEG-LDPC and tree codes

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.12. 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 23 confirm that Module $\mathcal{A}$ primarily resolves overlap and Module $\mathcal{B}$ primarily resolves parity inconsistency.

L.13. Inference visualizations

Figures 9–12 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 9 shows that CIDER progressively resolves ambiguity and converges to two distinct, globally consistent codewords. Figure 10 shows the generic MDD baseline, which often struggles to stabilize refinement under unsourced ambiguity. Figure 12 (no demixing, i.e., removing $\mathcal{A}$) frequently exhibits *duplicate-row collapse*, where both rows lock onto the same high-evidence symbols due to the lack of explicit row competition. Figure 11 (no parity-aware propagation, i.e., removing $\mathcal{B}$) often yields *row swapping/mismatched assembly* across slots, where the two rows repeatedly exchange roles during refinement.

L.14. 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 $\big(\mathrm{CIDER} + \mathrm{PRISM}\big)_K$ models), 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

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

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 25, 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 24 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 $K_{\mathrm{tot}} \le 60$) 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 always enable PRISM for every supported bin size:

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

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 13.

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 $\big(\mathrm{CIDER}+\mathrm{PRISM}\big)_{K_\chi}$ when $K_\chi \le K_{\max}$. 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 24 reports results with a fixed bin count ($\zeta = 25$), and Table 25 reports results with ζ scaled with K_{tot} to maintain average per-bin load $\mathbb{E}[K_\chi] \approx 4$.

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 are capped at 50 BP iterations with early exit on convergence. Top-J exhaustive search did not finish within 24 hours for $K \geq 4$.

Method	$K = 3$			$K = 4$			$K = 5$		
	SER	CER	Time	SER	CER	Time	SER	CER	Time
CIDER	**0.0006**	**0.0044**	**6.94**	**0.0015**	**0.0058**	**14.80**	**0.0048**	**0.0141**	**23.28**
FFT-BP	0.0036	0.0063	98.62	0.0380	0.0573	236.05	0.0393	0.0738	335.22
Top-J (Top 3)	0.1889	0.2850	9470.14	–	–	–	–	–	–
Top-J (Top 4)	0.0812	0.1013	316925.62	–	–	–	–	–	–

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**

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 $K_{\text{tot}} \gg K_{\max}$ users per frame while keeping the *worst-case per-bin* decoding cost bounded (by choosing ζ and enforcing the overflow rule).

Table 20. Sensitivity of CIDER to symbol-wise soft detector mismatch on Tiny $(Q, L) = (64, 12)$, $K = 2$, without retraining.

Mismatch type	**Condition**	**SER (↓)**	**CER (↓)**
None (matched)	SNR $= -0.79$ dB ($E_b/N_0 = 10$ dB), $I_{\mathrm{AMP}} = 20$	0.0011	0.0073
Reduced AMP iters	SNR $= -0.79$ dB ($E_b/N_0 = 10$ dB), $I_{\mathrm{AMP}} = 10$	0.0009	0.0066
Reduced AMP iters	SNR $= -0.79$ dB ($E_b/N_0 = 10$ dB), $I_{\mathrm{AMP}} = 5$	0.0009	0.0065
SNR mismatch	SNR $= -2.79$ dB ($E_b/N_0 = 8$ dB), $I_{\mathrm{AMP}} = 20$	0.0063	0.0318
SNR mismatch	SNR $= +1.21$ dB ($E_b/N_0 = 12$ dB), $I_{\mathrm{AMP}} = 20$	0.0008	0.0054
SNR mismatch	SNR $= -4.79$ dB ($E_b/N_0 = 6$ dB), $I_{\mathrm{AMP}} = 20$	0.1020	0.3087

Table 21. 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	87.20 ms
FFT-BP	0.0015	0.0062	7.99 ms
Top-2	0.0473	0.0497	73.77 ms
Top-3	0.0118	0.0120	9683.42 ms

Table 22. Tree-code comparison on Tiny $(Q, L) = (64, 12)$, $K = 2$. Lower is better.

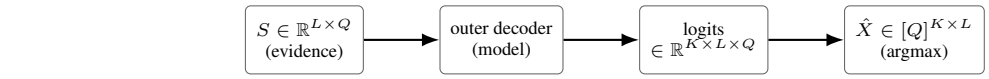
Method	**SER (↓)**	**CER (↓)**	**Time/sample (↓)**
CIDER	**0.0001**	**0.0007**	**1.08 ms**
SIC-BP	0.0017	0.0065	84.80 ms
FFT-BP	0.0017	0.0065	7.78 ms
Tree-code stitching decoder	0.0474	0.0490	1.20 ms
Top-2	0.0474	0.0490	75.17 ms

Table 23. Failure-mode diagnostics on Tiny $(Q, L) = (64, 12)$, $K = 2$. Rates are percentages over the test set.

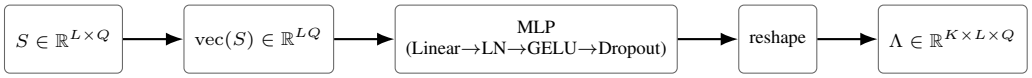
Model	**Slot-overlap rate**	**Parity-violation rate**	**SER**
MDD	63.5%	99.9%	0.4201
CIDER w/o $\mathcal{A}$	42.6%	99.9%	0.4127
CIDER w/o $\mathcal{B}$	2.9%	100.0%	0.3991
CIDER	**1.5%**	**0.7%**	**0.0011**

Table 24. 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.

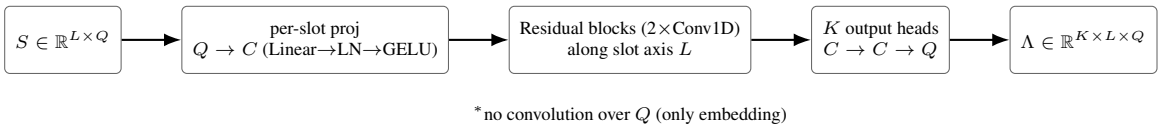
K_{tot}	**SER (↓)**	**CER (↓)**
10	0.0064	0.0417
20	0.0043	0.0398
30	0.0036	0.0369
40	0.0030	0.0258
50	0.0030	0.0207
60	0.0055	0.0189
70	0.0095	0.0205
80	0.0174	0.0278
90	0.0301	0.0394
100	0.0525	0.0619



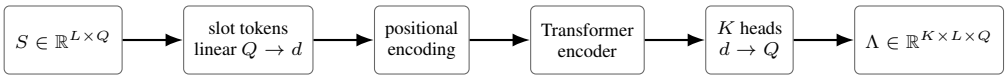
(a) Shared I/O interface ($S \mapsto \hat{X}$).



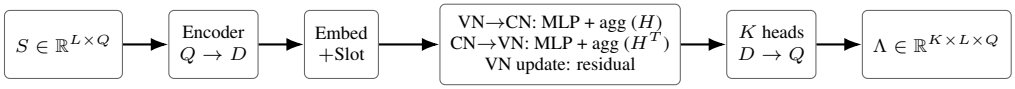
(b) MLP baseline.



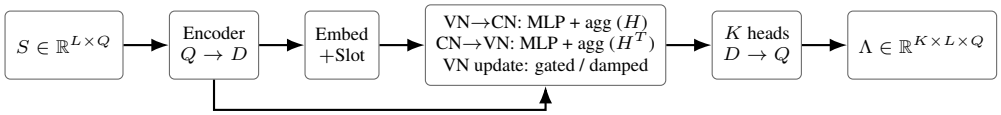
(c) CNN baseline.



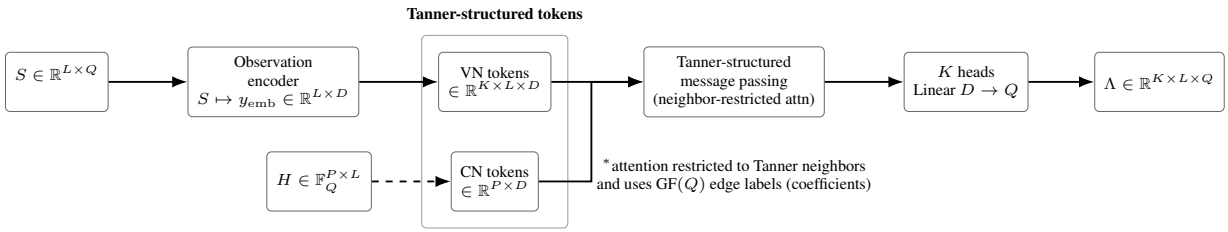
(d) Transformer baseline.



(e) GNN baseline.



(f) NBP baseline.

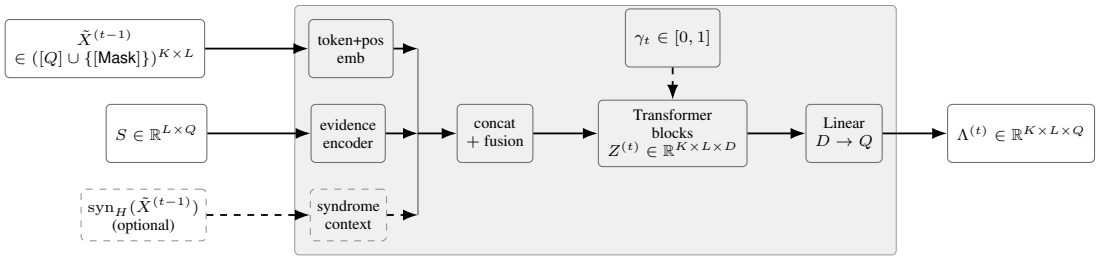


(g) TA baseline.

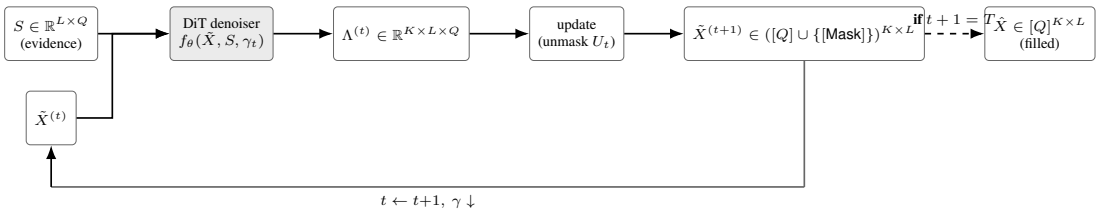
Figure 6. Architectures of compared multiuser decoders: the shared evidence-to-grid interface ($S \mapsto \hat{X}$), followed by one-shot baselines (MLP/CNN/Transformer) and Tanner-graph-aware baselines (GNN/NBP/TA). All methods consume the same evidence heatmap S and output $K \times L$ symbol grids.

Table 25. 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

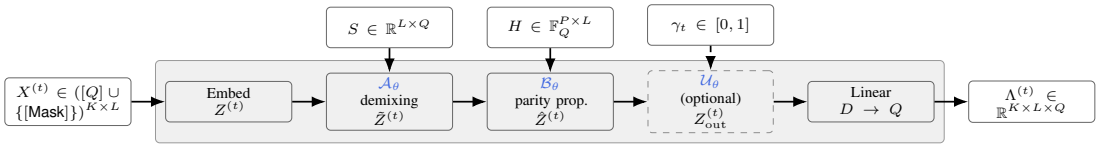


(a) MDD denoiser: the per-step conditional predictor.

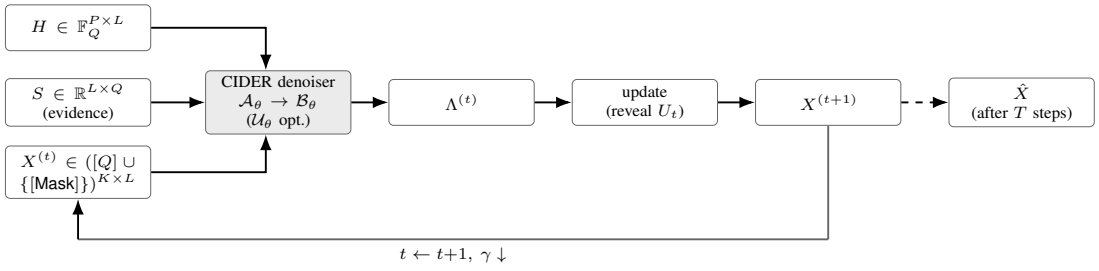


(b) Masked-diffusion refinement loop with the MDD denoiser.

Figure 7. MDD baseline: a generic masked-diffusion multiuser decoder instantiated with a standard denoiser.



(a) CIDER denoiser (per-step): structured modules and conditioning.



(b) Masked-diffusion refinement loop with the CIDER denoiser.

Figure 8. CIDER detailed view: denoiser internals and the fixed-step refinement loop.

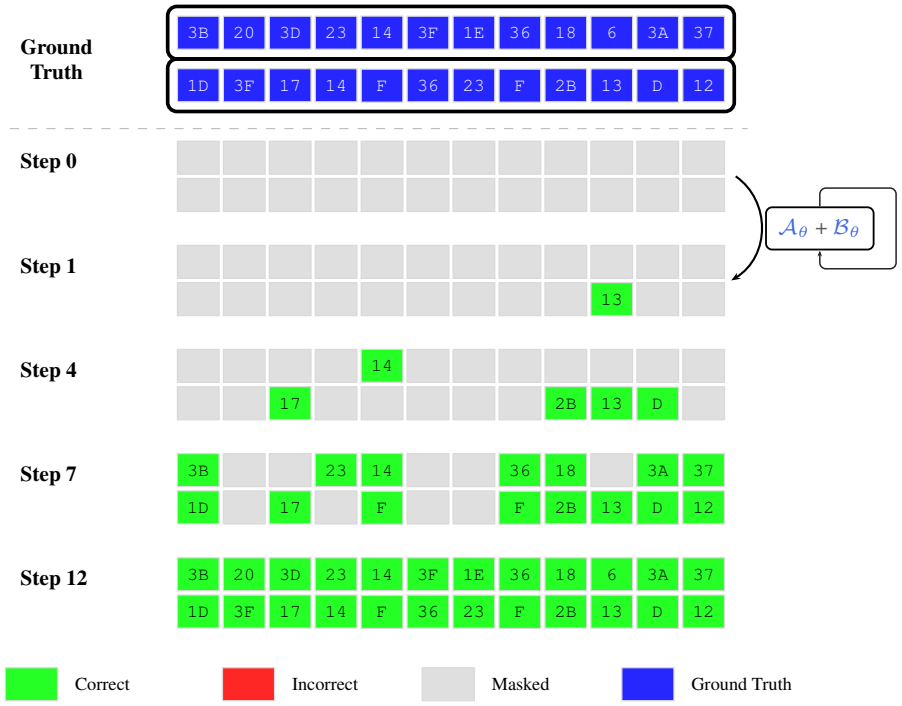


Figure 9. CIDER iterative decoding (full figure)

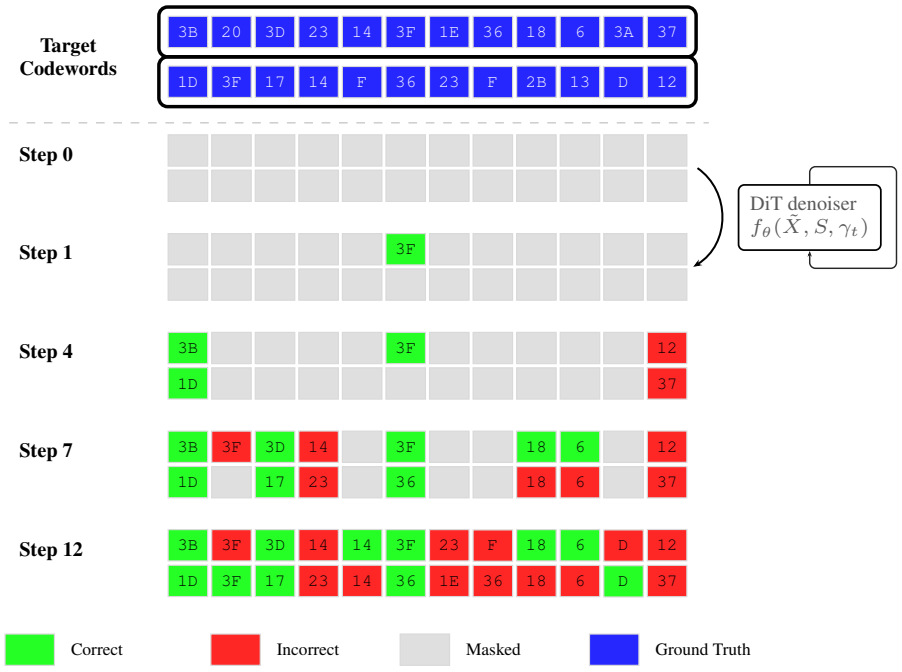


Figure 10. MDD iterative decoding

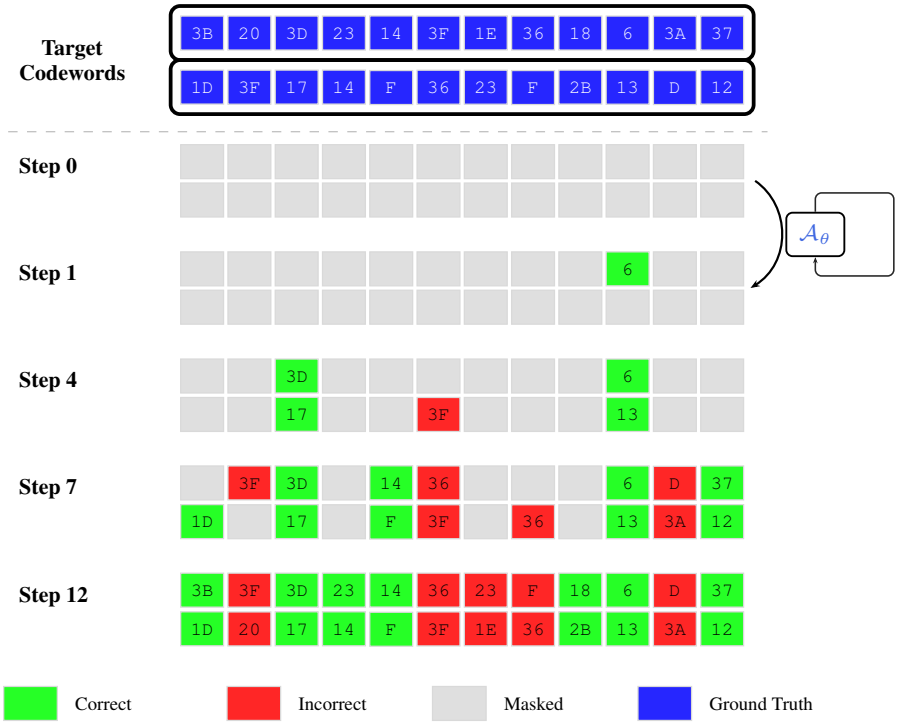


Figure 11. No parity-aware prop. ($\mathcal{A}$)

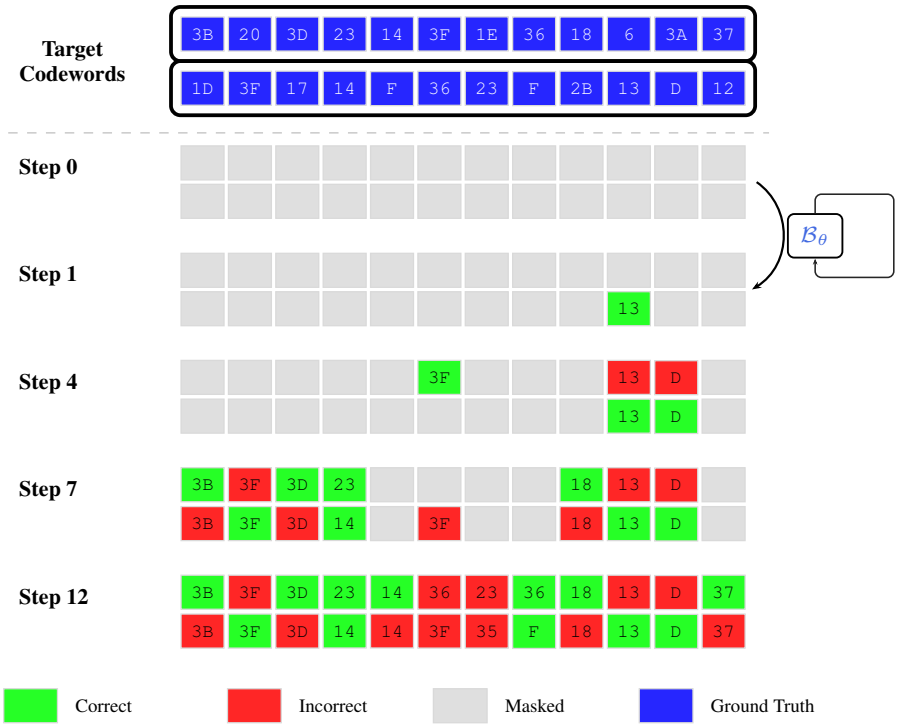


Figure 12. No demixing ($\mathcal{B}$)

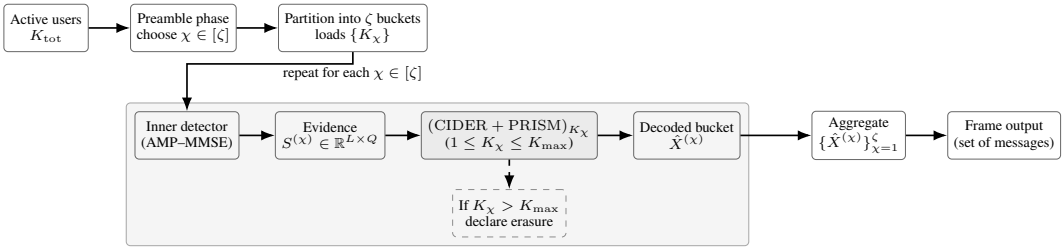


Figure 13. Protocol-level scalability wrapper: a two-step random-access protocol partitions K_{tot} active users into ζ preamble bins; each bin becomes an independent URA instance decoded by $(\mathrm{CIDER} + \mathrm{PRISM})_{K_\chi}$. Bins with $K_\chi > K_{\max}$ are declared failures (erasures).