# QUPEGGIO: QUANTUM ERROR-CORRECTING CODE DISCOVERY VIA AUTOREGRESSIVE GENERATION AND DIFFUSION REPAIR

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

## ABSTRACT

Quantum error-correcting (QEC) code discovery is essential for fault-tolerant quantum computation (FTQC), yet it remains largely driven by expert-designed constructions. Existing automated approaches based on reinforcement learning offer a promising alternative, but require enormous numbers of increasingly expensive code evaluations and have so far been limited to relatively small codes. Meanwhile, the field has accumulated millions of previously discovered QEC codes, raising a natural question: can this existing knowledge be reused to amortize code discovery across different design problems, rather than searching for every code from scratch? Moreover, can such learned knowledge help discover high-distance codes competitive with those already known?

We introduce a generate-and-repair framework that learns reusable construction patterns from existing QEC codes, combining global construction from learned priors with targeted local refinement that preserves useful existing structure. An autoregressive model generates code constructions across different numbers of physical and logical qubits, while a diffusion model selectively repairs parts of promising constructions that limit their code performance. Across large and diverse code specifications, our framework generalizes to sparsely represented and unseen settings, and discovers SAT-certified codes whose distances extend beyond the frontier represented in the training data. These results show that generative modeling can amortize QEC code discovery across design problems while continuing to improve upon the constructions from which it learns.

## 1 INTRODUCTION

Quantum error correction (QEC) is essential for realizing fault-tolerant quantum computation (FTQC) Shor (1996); Terhal (2015); Aharonov and Ben-Or (1997); Knill et al. (1998). Since the emergence of the first QEC codes in the 1990s Shor (1995); Steane (1996a); Gottesman (1996); Calderbank and Shor (1996), the field has progressed from early schemes encoding a single logical qubit into $\sim$10 physical qubits Shor (1995); Steane (1996a) to modern constructions encoding multiple logical qubits into $\sim$100 physical qubits Bravyi et al. (2024); Xu et al. (2024). Achieving this progress has relied heavily on expert-driven discovery, drawing on specialized domain knowledge to identify promising mathematical structures and develop them into effective constructions Bravyi et al. (2024); Tillich and Zémor (2013). However, this manual, case-by-case approach limits both the breadth and pace of code discovery, motivating automated QEC code discovery.

Recent work has explored reinforcement learning (RL)-based search to automate QEC code discovery Su et al. (2023); Olle et al. (2024). However, this approach has so far been limited to the $\sim$20-physical-qubit regime, leaving a substantial gap to the $\sim$100-physical-qubit regime of recent QEC constructions Olle et al. (2024). A key bottleneck is the large number of code evaluations required during RL search, with millions to billions of evaluations required to construct every target code Olle et al. (2024). Moreover, the cost of each code evaluation grows combinatorially as codes scale Vardy (1997), making the overall search computationally prohibitive for larger codes.

Meanwhile, the field has accumulated a rich collection of QEC codes Bravyi et al. (2024); Albert and Faist (2026), providing substantial prior knowledge about code constructions. Rather than searching



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

for each code from scratch, can we learn from existing codes and reuse that knowledge across future design problems, amortizing the effort of discovery? More ambitiously, could such knowledge guide the construction of codes with stronger code performance than those already known?

Generative modeling Vaswani et al. (2017); Austin et al. (2021), with its ability to learn from existing data and produce novel structured designs, offers a promising way toward realizing these ideas, yet applying it to QEC code discovery is nontrivial. First, codes must be represented in a form expressive enough to capture diverse constructions while ensuring that generated codes are valid. Second, the learned construction principles must generalize across codes with different numbers of physical and logical qubits, rather than remaining specific to a particular code size. Third, generating stronger codes requires learning what makes a code perform well, rather than merely imitating existing codes.

Even with these challenges addressed, relying solely on direct generation is insufficient for pushing performance further. For an already promising code, the remaining performance bottlenecks often stem from a small number of local construction choices. Simply regenerating the entire code to address these local limitations can undo useful structures and degrade the performance gains already achieved. A more effective strategy is therefore to perform precise refinement, making targeted local improvements while preserving the code's existing strengths.

Here, we introduce a generate-and-repair framework for QEC code discovery. For code generation, we represent each QEC code as an encoding circuit, which preserves code validity by design and recasts code construction as autoregressive sequence generation over gate operations. We then condition the model on the numbers of physical and logical qubits, enabling knowledge reuse across code sizes. Finally, we use code distance, a direct measure of error-correction capability, as a quality-conditioning signal, allowing the model to learn how construction choices relate to code performance. This enables performance knowledge from previously discovered codes to be reused across future design problems.

For code refinement, we turn global evaluation feedback into localized guidance. Specifically, we trace minimum-weight logical errors back through the encoding circuit to locate the gate operations that limit code performance. We then use a diffusion model to selectively repair these operations conditioning on the rest of the construction that already works well. Compared with the autoregressive modeling used for code generation, diffusion is naturally suited to localized conditional regeneration without imposing a left-to-right reconstruction order.

In summary, our contributions are:

• **Amortized QEC code discovery.** We formulate QEC code discovery as conditional generative modeling, enabling construction knowledge to be reused across different code specifications.

• **Error-guided repair.** We introduce a diagnosis-and-repair mechanism that uses minimum-weight logical errors to identify vulnerable circuit operations and selectively refine them with diffusion.

• **Large-scale discovery of strong QEC codes.** Across large and diverse code specifications, our framework generalizes to sparsely represented and unseen settings, and discovers SAT-certified codes whose distances extend beyond the frontier represented in the training data.

## 2 Background

**Quantum error correction.** A QEC code is characterized by parameters $[[n, k, d]]$ Gottesman (1997). As shown in Fig. 1(a), $n$ is the number of physical qubits and $k$ is the number of logical qubits encoded by the code. The ratio $k/n$, known as the encoding rate, measures how efficiently physical qubits are used to encode logical quantum information Gottesman (2013). The code distance $d$ provides a standard measure of a code's ability to protect against physical errors. Specifically, the code distance $d$ is the minimum number of physical-qubit errors that can produce an undetectable logical error. Thus, a distance-$d$ code can detect any error pattern occurring on fewer than $d$ physical qubits and correct any error pattern occurring on up to $\lfloor (d-1)/2 \rfloor$ physical qubits Gottesman (1997). Among QEC codes, Calderbank–Shor–Steane (CSS) codes are particularly important, as their underlying structure enables efficient error-correction procedures Gottesman (1997); Steane (1996b). Notably, they include some of the most prominent code families for FTQC, such as surface codes and BB codes Bravyi et al. (2024); Bravyi and Kitaev (1998). Developing CSS codes with stronger error-correction capability is therefore an important goal for advancing FTQC.

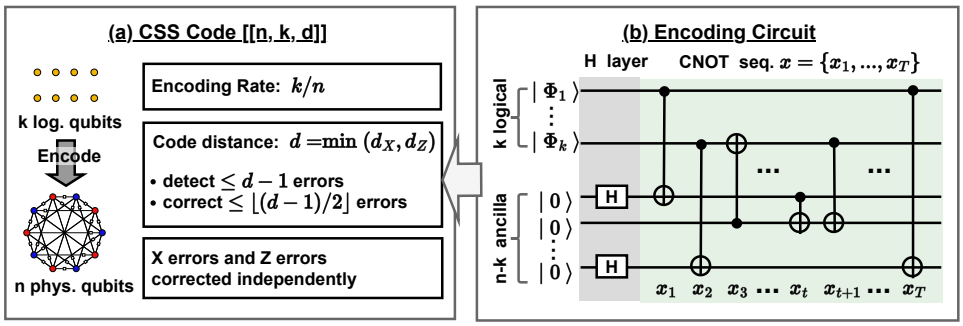



Figure 1: Encoding-circuit representation of a CSS code. Given $n$ physical qubits and $k$ logical qubits, a Hadamard layer followed by a CNOT sequence $x_{1:T}$ defines the code, while its distance $d$ is obtained by code-distance evaluation.

**Encoding-circuit representation.** An $[[n, k, d]]$ CSS code can be represented by an encoding circuit (Fig. 1(b)) that maps $k$ logical input qubits, together with $n-k$ ancilla qubits initialized in $|0\rangle$, to an encoded state on $n$ physical qubits Gottesman (1997); Cleve and Gottesman (1997). This circuit consists of an initial layer of Hadamard gates followed by a sequence of CNOT gates Gottesman (1997). Specifically, the Hadamard pattern determines the division between X- and Z-type stabilizer generators and is typically balanced Calderbank and Shor (1996); Gottesman (1997), reflecting the symmetric treatment of $X$- and $Z$-type errors. The subsequent CNOT sequence determines the detailed structure of the encoding circuit, including qubit interactions, control–target roles, and the ordering of these interactions. Together, the Hadamard pattern and CNOT sequence specify the code and its error-correction properties. An important property of this representation is that any valid encoding circuit naturally corresponds to a valid CSS code, avoiding the need to separately enforce code-validity constraints.

**Code distance evaluation.** Assessing a QEC code requires searching for minimum-weight logical errors and estimating its code distance Gottesman (1997). Computing the exact distance of a CSS code is NP-hard Vardy (1997), with straightforward exact search scaling exponentially in the number of physical qubits. In practice, randomized information-set methods are therefore commonly used to efficiently estimate code distance Pryadko et al. (2023). Even with such methods, a single evaluation can take on the order of seconds to minutes at practically relevant code sizes, making repeated code distance evaluations a substantial computational cost in automated code discovery.

**Generative models.** Generative models have been increasingly used to learn reusable distributions over structured scientific objects, including molecules, proteins, and circuits Gómez-Bombarelli et al. (2018); Segler et al. (2018); Gao et al. (2025). Autoregressive models represent such objects as sequences of decisions, generating each component conditioned on the preceding construction, and are therefore well suited to capturing dependencies across successive design steps Segler et al. (2018); Flam-Shepherd et al. (2022); Bagal et al. (2021). Diffusion models, in contrast, learn to reconstruct corrupted or masked components conditioned on the surrounding structure, naturally supporting conditional regeneration and localized modification Trippe et al. (2022); Chang et al. (2022). Together, these two modeling strategies offer complementary capabilities for structured design and are promising for QEC code discovery.

## 3 METHOD

Our framework follows a generate-and-repair pipeline, as shown in Fig. 2. Given a target specification $[[n, k, d^\star]]$, a conditioned autoregressive model first generates a complete encoding circuit (Fig. 2(a)), which is then evaluated to obtain its realized distance and minimum-weight logical errors (Fig. 2(b)). We backtrack these errors through the circuit to identify gate operations most responsible for the observed low-weight failure modes (Fig. 2(c)). The resulting gate-level scores define localized repair regions, which are regenerated by a bidirectional discrete diffusion model while the remaining circuit is kept fixed as context (Fig. 2(d)). The repaired circuit is then re-evaluated, combining global autoregressive generation with targeted local repair.

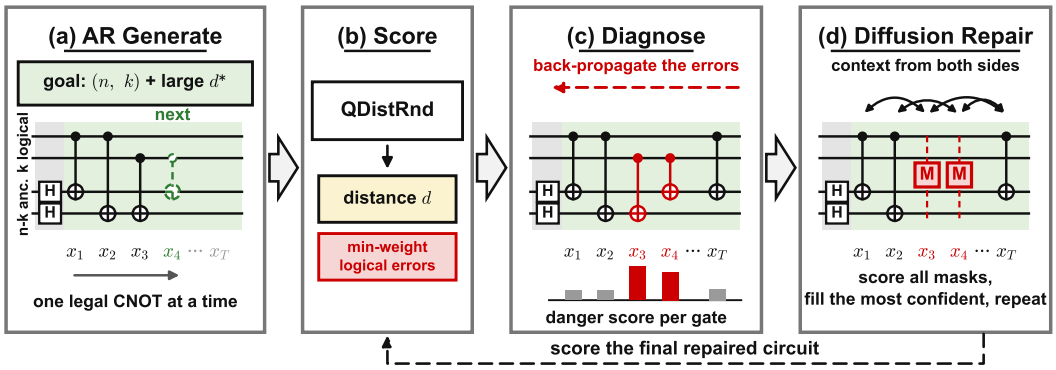



Figure 2: **The generate-and-repair pipeline.** (a) An AR model generates the encoding circuit gate by gate. (b) QDistRnd evaluates the generated code and identifies its minimum-weight logical errors. (c) These errors are back-propagated through the circuit to assign a danger score to each gate. (d) The most vulnerable gates are masked and regenerated by a discrete diffusion model.

### 3.1 Autoregressive Generator

Our generator learns a conditional distribution over QEC encoding circuits from existing code constructions, enabling a single model to generate candidate codes across different design specifications.

**Encoding-circuit representation.** We represent a QEC code by its encoding circuit as a sequence $C = (x_1, x_2, \ldots, x_T)$, where each token $x_t$ specifies a CNOT operation, including its control and target qubits, and $T$ is the total number of CNOT operations. The sequence order specifies how these operations are composed, so the effect of each gate depends on the circuit prefix $x_{1:t-1}$. We therefore use this sequence representation as the input representation for autoregressive modeling.

**Code-parameter-conditioned generation.** Given a desired code specification $[[n, k, d^\star]]$, we model the conditional probability distribution $p_\theta(C \mid n, k, d^\star) = \prod_{t=1}^{T} p_\theta(x_{t+1} \mid x_{1:t}, n, k, d^\star)$, where $n$ and $k$ denote the numbers of physical and logical qubits, respectively, and $d^\star$ denotes a target code distance. The model predicts each gate conditioned on both the previously generated circuit prefix and the target specification. While $(n, k)$ determines the required code configuration, $d^\star$ acts as a soft quality-conditioning signal rather than a hard constraint. A circuit generated with target distance $d^\star$ is therefore not required to achieve distance $d^\star$. Instead, distance conditioning allows the model to associate different circuit structures with different levels of error-correction capability. Empirically, for fixed $(n, k)$, larger values of $d^\star$ increase the likelihood of sampling candidates with higher realized code distance. The actual distance $d(C)$ is determined only through code evaluation.

**Training.** We parameterize the conditional distribution $p_\theta$ with a causal transformer. Each gate operation $x_t$ is represented by a learned token embedding together with a positional embedding. The target code specification $[[n, k, d]]$ is encoded using learned embeddings and combined into a global conditioning vector that is shared across all circuit positions, allowing every gate prediction to depend on the desired code specification.

We train the generator on a collection $\mathcal{D}$ of existing QEC codes represented by their encoding circuits, with details of the training-data provided in Appendix A. Each training circuit $C$ is paired with its corresponding code parameters $(n_C, k_C, d_C)$. Using teacher forcing, the model predicts each gate $x_{t+1}$ conditioned on the ground-truth circuit prefix $x_{1:t}$ and the associated code specification. We optimize the autoregressive negative log-likelihood objective $\mathcal{L}_{\text{gen}} = -\sum_{C \in \mathcal{D}} \sum_{t=1}^{T_C} \log p_\theta \left(x_{t+1} \mid x_{1:t}, n_C, k_C, d_C\right)$. A causal attention mask prevents the model from accessing future gate operations, while an end-of-sequence (EOS) token indicates circuit completion and enables variable-length generation. Training jointly across codes with different $[[n, k, d]]$ values allows the model to capture structural regularities shared across code-design tasks and to associate these structures with different levels of error-correction capability.

**Generation.** At inference time, we specify $[[n, k, d^\star]]$ and sample complete encoding circuits from $p_\theta(C \mid n, k, d^\star)$. Because the learned distribution is shared across code specifications, a single

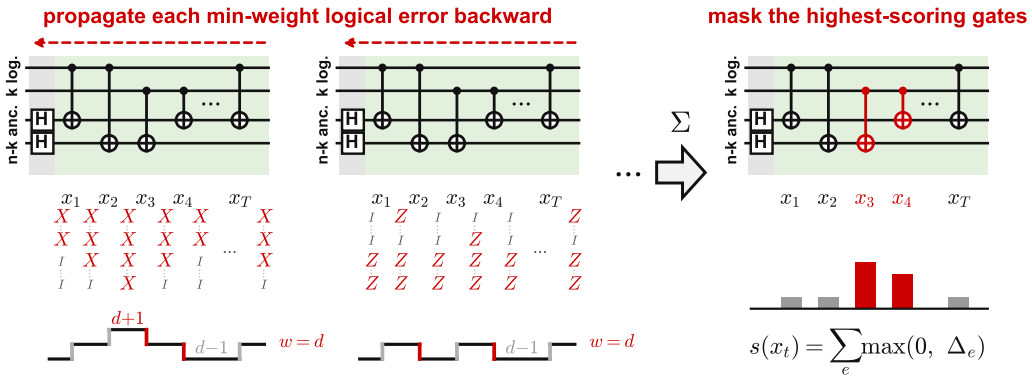



Figure 3: **Error guided diagnosis.** Each minimum-weight logical error $e$ is propagated backward through the encoding circuit, with the $X/Z/I$ entries showing the evolving Pauli error and the curves showing its weight along the circuit. For each gate $x_t$, $\Delta_e$ denotes the change in error weight when $e$ is propagated backward through $x_t$. Positive contributions are aggregated across errors as $s(x_t) = \sum_e \max(0, \Delta_e)$, and the high-scoring gates are masked for repair.

model generates candidates for different $(n, k)$ settings, amortizing code construction across design tasks. We choose a high target distance $d^\star$ to push generation beyond the code distances represented by the training data at the target scale. Even when such distances are rare or absent for the target $(n, k)$ in the training data, the same distance values can occur at other specifications, for example with larger $n$ or smaller $k$. Conditioning on $d^\star$ therefore allows the model to learn circuit structures associated with high distance across the training distribution and reuse these patterns when generating for the target specification. Since $d^\star$ is a soft conditioning signal rather than a hard constraint, the generated circuit is not required to achieve $d^\star$ exactly. We therefore evaluate each candidate to obtain its realized error-correction performance and apply the error-guided repair procedure in Sec. 3.2 and 3.3 to candidates that remain below the desired performance.

### 3.2 Error-guided Diagnosis

The autoregressive generator provides promising global code constructions, but a generated candidate still contains a small number of gate operations that limit their error-correction capability. The realized distance $d(C)$ measures overall code quality but does not indicate where the circuit should be modified. We therefore exploit minimum-weight logical errors to identify gate operations associated with these failure modes.

**Backtracing minimum-weight logical errors.** Given an encoding circuit $C = (x_1, \ldots, x_T)$, let $\mathcal{E}_{\min}(C)$ denote a set of minimum-weight logical errors identified by the evaluator. The code distance $d(C)$ is the minimum Pauli weight of a nontrivial logical error, so each $e \in \mathcal{E}_{\min}(C)$ satisfies $\mathrm{wt}(e) = d(C)$. Rather than using these errors only to determine the code distance, we backtrace them through the encoding circuit to identify the gate operations that contribute to their low-weight structure, as illustrated in Fig. 3.

For a logical Pauli error $e \in \mathcal{E}_{\min}(C)$, let $e^{(t)}$ denote the same error propagated backward to the circuit position after gate $x_t$. Since $T$ is the total number of gates, $e^{(T)} = e$ corresponds to the error at the circuit output. We then propagate it backward recursively as $e^{(t-1)} = x_t^\dagger e^{(t)} x_t, t = T, \ldots, 1$. Because Clifford gates map Pauli errors to Pauli errors under conjugation, each $e^{(t)}$ can be represented as a Pauli string over the physical qubits and propagated exactly through the circuit. We define the error weight at position $t$ as $w_e^{(t)} = \mathrm{wt}\big(e^{(t)}\big)$, so the trajectory $\{w_e^{(t)}\}_{t=0}^{T}$ shows where the support of the logical error changes as it is traced backward through the circuit.

**Gate-level diagnosis.** We use changes in the backtraced error weight to quantify the involvement of each gate in a minimum-weight logical error. For an error $e$, we define the contribution of gate $x_t$ as $\Delta_e(x_t) = w_e^{(t-1)} - w_e^{(t)}$, such that a positive $\Delta_e(x_t)$ indicates that crossing $x_t$ during backtracing increases the error weight. Intuitively, such a gate corresponds to a forward circuit operation that concentrates the support of the observed low-weight logical error, and is therefore a promising

location to modify in order to disrupt that failure mode. A single minimum-weight logical error, however, may identify gates that are specific to only one failure mode. We therefore aggregate evidence across all minimum-weight logical errors: $s(x_t) = \sum_{e \in \mathcal{E}_{\min}(C)} \max(\Delta_e(x_t), 0)$. The score $s(x_t)$ measures how consistently and strongly gate $x_t$ participates in the forward weight reductions of the current minimum-weight logical errors. Gates with large scores are implicated by multiple failure modes and are higher-priority targets for repair.

**Selecting gates for repair.** The gate scores $s(x_t)$ indicate how strongly each gate is implicated in the current minimum-weight logical errors. We use Otsu's method to determine an adaptive threshold $s_{\text{thr}}$ from the score distribution and select $A = \{t \mid s(x_t) > s_{\text{thr}}\}$. The selected gates form the repair mask $M$. During repair, only gates in $M$ are regenerated, while all remaining gates are kept fixed and provide context to the diffusion model. This allows the model to modify the most vulnerable parts of the circuit while preserving its overall structure.

### 3.3 Diffusion Repair

The autoregressive model generates the full circuit, while repair modifies only selected gate positions. We therefore use a discrete diffusion model to regenerate these gates conditioned on the surrounding circuit and target code specification.

**Training.** We use the same gate-sequence representation $C = (x_1, \ldots, x_T)$ as in the autoregressive generator and parameterize the reverse process with a bidirectional transformer. We adopt an absorbing-mask discrete diffusion process, in which gate tokens are progressively replaced by a special mask token as the diffusion timestep increases and remain masked thereafter in the forward process. For each training circuit, we randomly sample a set of positions $M_{\text{train}}$ to corrupt, while all positions outside $M_{\text{train}}$ remain unchanged. Let $C^{(\tau)}$ denote the resulting partially corrupted circuit at diffusion step $\tau$. The model learns the conditional reverse process $p_\phi\left(C^{(\tau-1)}_{M_{\text{train}}} \mid C^{(\tau)}, n, k, d, \tau\right)$, where the uncorrupted gates provide fixed bidirectional context for reconstructing the corrupted positions. The code parameters $(n, k, d)$ are represented by learned embeddings and combined with the diffusion-timestep embedding as a global conditioning signal shared across circuit positions. Training over random corruption regions and diffusion timesteps teaches the model to reconstruct local circuit structure from its surrounding context.

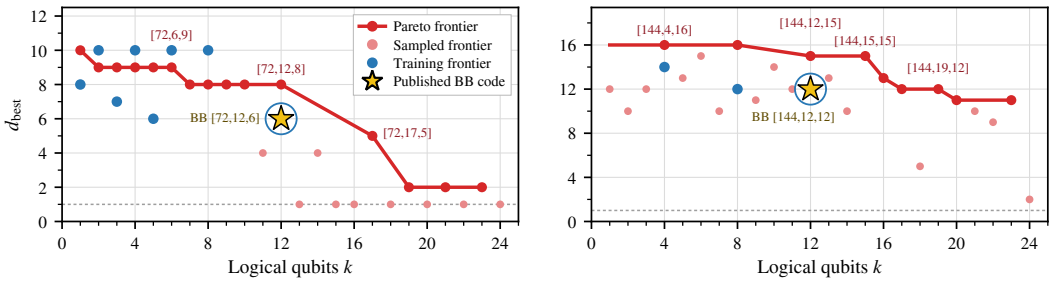
**Error-guided repair.** At inference time, we replace the random training mask with the repair mask $M$ obtained from the error-guided diagnosis in Sec. 3.2. Only positions in $M$ are corrupted and regenerated, while all remaining gates are clamped to their original values throughout the reverse process, $x_t^{(\tau)} = x_t, t \notin M$. Starting from the corrupted masked positions, the model iteratively denoises them while conditioning on the fixed surrounding circuit and the specification $[[n, k, d^\star]]$. Because the transformer is bidirectional, the selected positions can be updated using both their surrounding circuit context and the current states of other positions being repaired. As with generation, $d^\star$ serves as a conditioning signal rather than a hard constraint, and the distance $d(C')$ is obtained by code evaluation. We then compare $d(C')$ with $d(C)$ and retain the circuit with the larger distance.

## 4 Evaluation

### 4.1 Experiment Setup

**Evaluation tasks.** We evaluate high-distance QEC code discovery for $n \in \{72, 144\}$ physical qubits, with published BB codes Bravyi et al. (2024) included as reference points at these code sizes. For each $n$, we sweep the number of logical qubits $k$ across different encoding rates, treating each $n, k$ as a separate design task. Unless otherwise specified, ablations use $n = 144$ and $k = 15$, a large-scale intermediate-rate setting with sufficient headroom to distinguish different generation and repair strategies.

**Training setup.** We parameterize the autoregressive generator with a causal transformer and the diffusion repair model with a bidirectional transformer. The models are trained on $\sim$1.4 million CSS codes spanning eleven code families Shor (1995); Steane (1996a); Bravyi et al. (2024); Tillich and Zémor (2013); Bravyi and Kitaev (1998); Kovalev and Pryadko (2013); Panteleev and Kalachev




**(a)** $n = 72$, $N_{\text{eval}} = 600$ **(b)** $n = 144$, $N_{\text{eval}} = 1200$

Figure 4: **Large-scale QEC code discovery across code specifications.** The per-task QEC-evaluation budget is 600 for $n = 72$ and 1200 for $n = 144$. Pink markers show the best sampled code for each target specification, red markers show the Pareto frontier, blue markers show the training-data frontier, and stars indicate published BB codes as reference points.

(2021); Bombin and Martin-Delgado (2006); Kitaev (2003); Grassl and Beth (1999). Full details of the training data, model architectures are provided in Appendix A and B.

**Evaluation metric.** Our primary metric is the best code distance discovered within $N_{\text{eval}}$ evaluations. We report $d_{\text{best}} = \max_{i \le N_{\text{eval}}} d(C_i)$, where $C_i$ denotes the $i$-th evaluated candidate and $d(C_i)$ is its code distance estimated by QDistRnd Pryadko et al. (2023). For codes reported on the Pareto frontier, we recompute their code distances using an SAT solver Webster et al. (2026), which searches for a minimum-weight logical error and certifies its optimality. Thus, all reported Pareto-frontier distances are exact.

**Evaluation protocol.** We use the number of code-distance evaluations $N_{\text{eval}}$ as the common compute budget across methods. Unless otherwise specified, we use two repair rounds, following Sec. 4.4. Autoregressive-only generation requires one QEC evaluation per candidate, while generation with repair requires three, all counted toward the same evaluation budget. Code-distance evaluation is the dominant recurring cost for each target. Model training has a comparable overall computational cost, but is performed only once and shared across all code specifications. In contrast, RL-based search performs target-specific policy optimization, so its optimization cost is incurred separately for each $(n, k)$ setting. Detailed compute costs are provided in Appendix D.

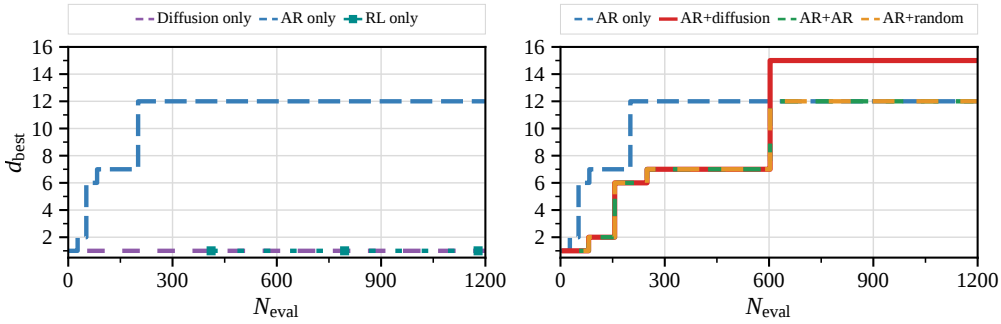
**Baselines.** For overall code discovery performance, we compare a diverse set of proposal strategies, including diffusion generation and the SOTA RL-based search baseline Olle et al. (2024). For repair, all methods start from the same autoregressively generated candidate codes, isolating the effect of the repair strategy. Results across additional random seeds are provided in Appendix F, showing consistent trends across independent runs.

### 4.2 Main Results

We evaluate our framework across different $(n, k)$. Fig. 4 shows the discovered $k$–$d$ frontiers for $n \in \{72, 144\}$ under a fixed QEC-evaluation budget, compared to those in the training data.

**Transfer across code specifications.** Our framework transfers across target code specifications, including $(n, k)$ configurations that are sparsely represented or absent from the training data. For $n = 72$, the training set covers only $33.3\%$ of the evaluated $k$ values, while for $n = 144$ the coverage drops to $12.5\%$. Nevertheless, the framework discovers high-distance codes across the remaining unseen specifications, demonstrating cross-specification reuse of learned construction patterns within the code distribution considered here.

**Discovery beyond the training frontier.** Beyond generalizing across specifications, the framework also discovers codes whose distances exceed the best performance represented in the training data. For $n = 72$, the discovered codes surpass the training-data frontier at several configurations, while for $n = 144$ they exceed it throughout the evaluated sweep. At $k = 12$, the published BB codes $[[72, 12, 6]]$ and $[[144, 12, 12]]$ already lie on the training-data frontier and capture the best code distance in the full training set at corresponding code sizes (Appendix A), providing useful reference points. At these settings, our framework discovers $[[72, 12, 8]]$ and $[[144, 12, 15]]$, demonstrating that learned construction knowledge can support code discovery beyond the performance




(a) Generation backbone (b) Repair mechanism

Figure 5: **Generation backbone and repair mechanism ablations.** Here, $n = 144$ and $k = 15$. We use two repair rounds by default, so methods with repair require three QEC evaluations per initial proposal. (a) Comparison of proposal backbones. (b) Comparison of repair mechanisms.

frontier represented in the training data. Appendix H further characterizes the discovered codes, which have moderately higher check weights than the corresponding BB constructions, reflecting a trade-off between sparsity and distance while retaining substantial sparse structure.

**Generalization beyond training constructions.** We further examine whether the discovered codes simply reproduce constructions already present in the training set. Using quantum weight enumerators (QWEs) to distinguish code classes Rains (1998), we find that the discovered high-performing codes include QWE classes not represented in the training data, as detailed in Appendix E. This provides further evidence that the model does not merely reproduce training constructions, but learns reusable structural patterns that can generate distinct high-performing codes.

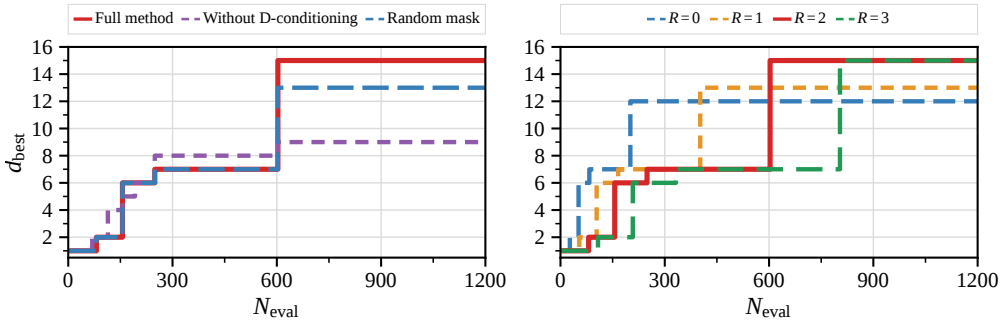
### 4.3 What Drives the Performance Gains?

**Strong autoregressive proposals.** We compare different proposal methods under the same QEC-evaluation budget. As shown in Fig. 5(a), autoregressive generation performs best, reaching a code distance of 12. Diffusion generation remains at low distance when generating complete circuits from scratch. In masking-based diffusion, gate operations are reconstructed from their surrounding context. When constructing a full circuit, little such context is available, making this setting less suitable for diffusion generation. The RL baseline also remains at $d = 1$ under 1200 QEC evaluations, which already corresponds to days of wall-clock search. The key challenge is the large per-target search space: at $n = 144$, each CNOT decision involves roughly $2 \times 10^4$ possible control–target choices, while reward is obtained only after evaluating a complete code. With current evaluation budget, this sparse feedback makes effective policy improvement difficult. In contrast, autoregressive generation amortizes discovery across tasks by learning reusable construction patterns from existing codes and directly proposing complete circuits. This allows construction patterns learned from existing codes to be reused when generating new candidates.

**Effective diffusion repair.** We next compare different repair strategies starting from the same autoregressive candidates and using the same repair regions. As shown in Fig. 5(b), diffusion repair improves the best code distance from 12 to 15, while random and autoregressive repair remain at 12. Random repair changes the selected gates without using patterns learned from existing codes, and therefore rarely produces coordinated improvements. Autoregressive repair can use learned construction patterns, but modifying an earlier part of the circuit requires regenerating subsequent operations, which can overwrite useful structure that already works well. In contrast, diffusion repair reconstructs the masked gates using bidirectional context while keeping the rest of the circuit fixed. This makes it well suited to local refinement, where most of the existing construction should be preserved and only a small number of gate operations need to be changed.

### 4.4 Ablation Studies

**Method-component ablation.** The purple dashed curve in Fig. 6(a) shows that removing $d^\star$ conditioning reduces the best code distance from 15 to 9. This confirms that $d^\star$ provides a useful quality signal, helping the model favor circuit structures with stronger error-correction capability. The blue




(a) Components (b) Repair rounds

Figure 6: **Method and repair-round ablations.** Here, $n = 144$ and $k = 15$. (a) Full method versus without $d^*$ conditioning and random masking. (b) Comparison of repair rounds $R = 0, 1, 2, 3$.

dashed curve in Fig. 6(a) further shows that replacing the error-guided repair mask with a random mask lowers the best code distance from 15 to 13. This shows that minimum-weight logical errors provide useful guidance on where to modify the circuit, allowing repair to focus on gate operations most involved in the current low-distance failure modes.

**Repair-round ablation.** Each repair round requires an additional QEC evaluation for the repaired candidates, so the total evaluation cost grows linearly with the number of repair rounds. Too few rounds may leave promising candidates insufficiently refined, while additional rounds increase cost and may provide diminishing returns. Fig. 6(b) studies this trade-off. The best code distance improves from 12 at $R = 0$ to 13 at $R = 1$ and 15 at $R = 2$, while a third round provides no further improvement. We therefore use $R = 2$ as the default setting, which provides the best balance between code improvement and QEC-evaluation cost.

## 5 Related Works

Prior work has approached automated QEC discovery as per-target search, with reinforcement learning providing a prominent example Su et al. (2023); Olle et al. (2024). Scaling such methods is challenging because the search space grows rapidly and candidate codes must be repeatedly evaluated. Concurrent work makes search more manageable by focusing on established code families, such as bivariate-bicycle and lifted-product codes, and optimizing their parameters or construction programs Yan et al. (2026); Cruz-Benito et al. (2026). This reduces the search space, but keeps discovery tied to a predefined family and still relies on extensive candidate evaluation. In these methods, the design specification is not fixed in advance, since changing the family parameters can also change the resulting code size and encoding rate. This makes a direct comparison of search efficiency less meaningful. For completeness, we compare the final codes at matched specifications in Appendix C, where our codes achieve higher distance than the best reported concurrent results. In contrast, our approach moves beyond search-based discovery by learning reusable construction patterns from existing codes and directly generating candidates across different code specifications with a single model.

## 6 Conclusion

We introduced a generate-and-repair framework for QEC code discovery that learns reusable construction patterns from existing codes and combines autoregressive generation with error-guided diffusion repair. By coupling global generation from a learned prior with targeted refinement of vulnerable circuit operations, the framework amortizes code discovery across diverse specifications while preserving useful existing structure. Across large-scale settings, it transfers to sparsely represented and unseen specifications and discovers SAT-certified codes whose distances extend beyond the frontier represented in the training data. These results demonstrate the potential of generative modeling to turn existing QEC constructions into reusable knowledge for discovering and iteratively improving new codes.

### AI USE STATEMENT

In this work, we used generative AI tools to assist with language editing, grammatical correction, and improving the clarity and readability of the manuscript. We did not use generative AI tools to generate research data, perform data analysis, interpret results, formulate scientific conclusions, or generate references, and other disclosure categories are not applicable to this work. Additionally, generative AI tools were used to suggest alternative wording and improve the presentation of selected passages. All AI-assisted content was carefully reviewed, edited, and verified by the authors to ensure accuracy, consistency with the underlying research, and appropriate scholarly attribution. No AI-generated references or factual claims were included without independent verification. We take full responsibility for the final content of this work, including any text, claims, or other materials produced with the aid of generative AI.

## REFERENCES

Peter W Shor. Fault-tolerant quantum computation. In *Proceedings of 37th conference on foundations of computer science*, pages 56–65. IEEE, 1996.

Barbara M Terhal. Quantum error correction for quantum memories. *Reviews of Modern Physics*, 87(2):307–346, 2015.

Dorit Aharonov and Michael Ben-Or. Fault-tolerant quantum computation with constant error. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 176–188, 1997.

Emanuel Knill, Raymond Laflamme, and Wojciech H Zurek. Resilient quantum computation. *Science*, 279(5349):342–345, 1998.

Peter W Shor. Scheme for reducing decoherence in quantum computer memory. *Physical review A*, 52(4):R2493, 1995.

Andrew M Steane. Error correcting codes in quantum theory. *Physical Review Letters*, 77(5):793, 1996a.

Daniel Gottesman. Class of quantum error-correcting codes saturating the quantum hamming bound. *Physical Review A*, 54(3):1862, 1996.

A Robert Calderbank and Peter W Shor. Good quantum error-correcting codes exist. *Physical Review A*, 54(2):1098, 1996.

Sergey Bravyi, Andrew W Cross, Jay M Gambetta, Dmitri Maslov, Patrick Rall, and Theodore J Yoder. High-threshold and low-overhead fault-tolerant quantum memory. *Nature*, 627(8005): 778–782, 2024.

Qian Xu, J Pablo Bonilla Ataides, Christopher A Pattison, Nithin Raveendran, Dolev Bluvstein, Jonathan Wurtz, Bane Vasić, Mikhail D Lukin, Liang Jiang, and Hengyun Zhou. Constant-overhead fault-tolerant quantum computation with reconfigurable atom arrays. *Nature Physics*, 20(7):1084–1090, 2024.

Jean-Pierre Tillich and Gilles Zémor. Quantum ldpc codes with positive rate and minimum distance proportional to the square root of the blocklength. *IEEE Transactions on Information Theory*, 60 (2):1193–1202, 2013.

Vincent Paul Su, C Cao, Hong-Ye Hu, Yariv Yanay, Charles Tahan, and Brian Swingle. Discovery of optimal quantum error correcting codes via reinforcement learning. *arXiv preprint arXiv:2305.06378*, 2023.

Jan Olle, Remmy Zen, Matteo Puviani, and Florian Marquardt. Simultaneous discovery of quantum error correction codes and encoders with a noise-aware reinforcement learning agent. *npj Quantum Information*, 10(1):126, 2024.

Alexander Vardy. The intractability of computing the minimum distance of a code. *IEEE Transactions on Information Theory*, 43(6):1757–1766, 1997.

Victor V Albert and Philippe Faist. The error correction zoo. 2026.

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

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021.

Daniel Gottesman. *Stabilizer codes and quantum error correction*. California Institute of Technology, 1997.

Daniel Gottesman. Fault-tolerant quantum computation with constant overhead. *arXiv preprint arXiv:1310.2984*, 2013.

Andrew Steane. Multiple-particle interference and quantum error correction. *Proceedings: Mathematical, Physical and Engineering Sciences*, pages 2551–2577, 1996b.

Sergey B Bravyi and A Yu Kitaev. Quantum codes on a lattice with boundary. *arXiv preprint quant-ph/9811052*, 1998.

Richard Cleve and Daniel Gottesman. Efficient computations of encodings for quantum error correction. *Physical Review A*, 56(1):76, 1997.

Leonid P Pryadko, Vadim A Shabashov, and Valerii K Kozin. Qdistrnd: A gap package for computing the distance of quantum error-correcting codes. *arXiv preprint arXiv:2308.15140*, 2023.

Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268, 2018.

Marwin HS Segler, Thierry Kogej, Christian Tyrchan, and Mark P Waller. Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS central science*, 4(1): 120–131, 2018.

Jian Gao, Weidong Cao, Junyi Yang, and Xuan Zhang. Analoggenie: A generative engine for automatic discovery of analog circuit topologies. In *International Conference on Learning Representations*, volume 2025, pages 96496–96514, 2025.

Daniel Flam-Shepherd, Kevin Zhu, and Alán Aspuru-Guzik. Language models can learn complex molecular distributions. *Nature Communications*, 13(1):3293, 2022.

Viraj Bagal, Rishal Aggarwal, PK Vinod, and U Deva Priyakumar. Molgpt: molecular generation using a transformer-decoder model. *Journal of chemical information and modeling*, 62(9):2064–2076, 2021.

Brian L Trippe, Jason Yim, Doug Tischer, David Baker, Tamara Broderick, Regina Barzilay, and Tommi Jaakkola. Diffusion probabilistic modeling of protein backbones in 3d for the motif-scaffolding problem. *arXiv preprint arXiv:2206.04119*, 2022.

Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman. Maskgit: Masked generative image transformer. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11305–11315. IEEE, 2022.

Alexey A Kovalev and Leonid P Pryadko. Quantum kronecker sum-product low-density parity-check codes with finite rate. *Physical Review A—Atomic, Molecular, and Optical Physics*, 88(1): 012311, 2013.

Pavel Panteleev and Gleb Kalachev. Degenerate quantum ldpc codes with good finite length performance. *Quantum*, 5:585, 2021.

Hector Bombin and Miguel Angel Martin-Delgado. Topological quantum distillation. *Physical review letters*, 97(18):180501, 2006.

A Yu Kitaev. Fault-tolerant quantum computation by anyons. *Annals of physics*, 303(1):2–30, 2003.

Markus Grassl and Thomas Beth. Quantum bch codes. *arXiv preprint quant-ph/9910060*, 1999.

Mark Webster, Abraham Jacob, and Oscar Higgott. Distance-finding algorithms for quantum codes and circuits. *arXiv preprint arXiv:2603.22532*, 2026.

Eric M Rains. Quantum weight enumerators. *IEEE Transactions on Information Theory*, 44(4): 1388–1394, 1998.

Ge Yan, Shanchuan Li, Pengyue Ma, Qixin Zhang, Pingchuan Ma, Jianping Wang, Min-Hsiu Hsieh, and Yuxuan Du. Omniqec: discovering practical quantum error-correcting codes by an ai scientist. *arXiv preprint arXiv:2607.25865*, 2026.

Juan Cruz-Benito, Andrew W Cross, David Kremer, and Ismael Faro. Evolutionary discovery of bivariate bicycle codes with llm-guided search. *arXiv preprint arXiv:2606.02418*, 2026.

Stefan Krastanov and Quantum Savory Contributors. Quantumsavory/quantumclifford.jl. https://github.com/QuantumSavory/QuantumClifford.jl, 2020.

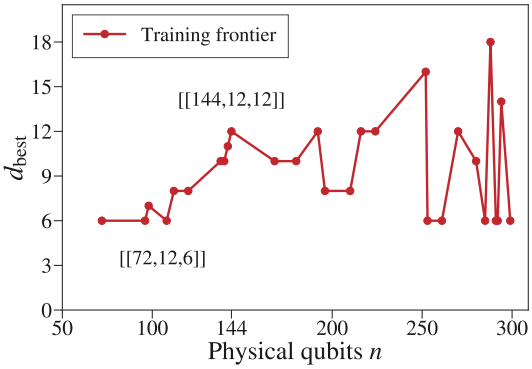
## A Training data Description

We collect training codes from a diverse set of existing QEC constructions provided by Quantum-Clifford.jl Albert and Faist (2026); Krastanov and Contributors (2020), covering eleven code families Shor (1995); Steane (1996a); Bravyi et al. (2024); Tillich and Zémor (2013); Bravyi and Kitaev (1998); Kovalev and Pryadko (2013); Panteleev and Kalachev (2021); Bombin and Martin-Delgado (2006); Kitaev (2003); Grassl and Beth (1999). Table 1 summarizes the resulting training-data distribution. In total, the dataset contains approximately 1.4 million training records. The resulting dataset spans a broad range of construction principles and provides the model with diverse structural patterns from which to learn reusable code-design knowledge.

Table 1: Training data distribution.

| Family | Training records |
|---|---|
| Color codes | 2461 |
| Surface codes | 225 |
| Bivariate bicycle (BB) | 608785 |
| Generalized bicycle (GB) | 361173 |
| Hypergraph product (HGP) | 119437 |
| Lifted product (LP) | 6748 |
| Toric codes | 510 |
| Quantum Hamming / Steane | 19 |
| Shor codes | 10 |
| Quantum BCH | 383 |
| Self-orthogonal CSS | 258940 |
| **Classified total** | **1358691** |

Figure 7: Best distance over all training codes (all 11 families) at k = 12.




Additionally, for $k = 12$, Fig. 7 shows the best code distance $d_{\text{best}}$ across the values of $n$ represented in the training data. The published BB codes $[[72, 12, 6]]$ and $[[144, 12, 12]]$ lie on the upper envelope of code distance across the full training set in their respective size ranges, providing strong reference points for demonstrating that our method extends beyond the training-data frontier.

## B Model Architectures

Table 2: Architectural configurations.

| Component | Autoregressive | Diffusion |
|---|---|---|
| Transformer layers | 24 | 12 |
| Hidden dimension | 768 | 512 |
| Attention heads | 8 | 8 |
| Feed-forward dimension | 3072 | 2048 |
| Attention pattern | Causal | Bidirectional |
| Normalization | Pre-LayerNorm | Pre-LayerNorm |
| Feed-forward activation | GELU | GELU |
| Dropout | 0 | 0.1 |
| Position embeddings | Learned absolute | Learned absolute |
| Token embedding | Factorized | Factorized |
| Output projection | Factorized | Factorized |
| Trainable parameters | 195M | 44M |

Both the autoregressive generator and the diffusion model are Transformer backbones with factored token embeddings and factored output projections, operating on the same circuit-token vocabulary and conditioned on the code parameters $[[n, k, d]]$, with the diffusion model additionally conditioned on the diffusion timestep. They differ in attention pattern and prediction objective: the autoregressive model predicts the next circuit token from a prefix, while the diffusion model reconstructs masked tokens from the surrounding circuit context. Table 2 summarizes both configurations.

Specifically, we embed the code parameters $n$, $k$, and $d$ independently and process the resulting embeddings with separate MLPs. Their outputs are then combined into a shared conditioning representation. At each Transformer layer, this representation is mapped to layer-specific FiLM parameters that modulate the hidden activations through feature-wise affine transformations. This per-layer conditioning allows the model to adapt its computation to different code specifications while sharing the same Transformer backbone across $[[n, k, d]]$ settings. For the diffusion model, the diffusion-timestep embedding is additionally incorporated into the conditioning representation before the layer-wise FiLM modulation.

## C Comparison with Concurrent Works

We further compare our discovered codes with those reported by concurrent automated QEC discovery methods Yan et al. (2026); Cruz-Benito et al. (2026). These methods perform discovery within predefined code families, whereas our framework learns construction patterns from a heterogeneous collection of existing codes and applies them across different code specifications. As summarized in Table 3, our method discovers codes with higher code distance at the same numbers of physical and logical qubits in the evaluated settings. The results suggest that reusing construction knowledge across different codes provides useful guidance for generating and refining candidates under new specifications. This allows the same framework to be applied across different design settings without requiring the search to be formulated for a specific code family.

Table 3: Comparison with concurrent methods at matched physical-qubit counts.

| Comparison with Cruz-Benito et al. (2026) | | | Comparison with Yan et al. (2026) | | |
|---|---|---|---|---|---|
| $(n, k)$ | Cruz-Benito et al. (2026) | Ours | $(n, k)$ | Yan et al. (2026) | Ours |
| $(72, 4)$ | $[[72, 4, 8]]$ | $[[\mathbf{72}, \mathbf{4}, \mathbf{9}]]$ | $(72, 12)$ | $[[72, 12, 6]]$ | $[[\mathbf{72}, \mathbf{12}, \mathbf{8}]]$ |
| $(72, 12)$ | $[[72, 12, 6]]$ | $[[\mathbf{72}, \mathbf{12}, \mathbf{8}]]$ | $(144, 8)$ | $[[144, 8, 12]]$ | $[[\mathbf{144}, \mathbf{8}, \mathbf{16}]]$ |
| $(144, 12)$ | $[[144, 12, 12]]$ | $[[\mathbf{144}, \mathbf{12}, \mathbf{15}]]$ | $(144, 12)$ | $[[144, 12, 12]]$ | $[[\mathbf{144}, \mathbf{12}, \mathbf{15}]]$ |

## D Compute Resources

All model-training experiments were run using $4\times$ NVIDIA H100 GPUs with 80 GB of memory each, while QEC code-distance evaluations were performed on AMD EPYC 9534 CPUs with 128 cores and 1.5 TiB of RAM. Training the autoregressive generator takes $\sim$4 days, whereas the diffusion repair model takes $\sim$1 day. Both models are trained only once and reused across all target code specifications. By contrast, QEC evaluation must be performed separately for each target $(n, k)$ and therefore constitutes the dominant target-specific computational cost. Specifically, 600 and 1200 QEC evaluations require $\sim$1 and 2 days of wall-clock time, respectively, matching the evaluation budgets used in the main experiments. We therefore use the number of QEC evaluations $N_{\text{eval}}$, as the primary measure of target-specific discovery cost.

## E Generalization to Unseen Code classes

To assess whether the high-performing codes discovered by our model simply reproduce code classes already present in the training data, we analyze the quantum weight enumerators (QWEs) of the discovered codes at the two matched BB-code settings considered in Fig. 4. Specifically, we consider codes with $k = 12$ that achieve at least the distance of the corresponding published BB code: $d \geq 6$ for $n = 72$ and $d \geq 12$ for $n = 144$. Following the QWE-based classification of quantum codes Rains (1998), we group codes with the same QWE into the same code class and compare the resulting classes against those represented in the training set.

For $n = 72$, among the discovered $[[72, 12, d]]$ codes with $d \geq 6$, we identify 187 QWE-defined classes that are not represented in the training data. For $n = 144$, among the discovered $[[144, 12, d]]$ codes with $d \geq 12$, we identify 52 previously unseen classes. Thus, even within the high-performing regime matching or exceeding the corresponding BB-code distances, the model discovers code

classes absent from the training data, providing further evidence that its performance gains do not arise from simply reproducing known training constructions.

## F RESULTS ON MULTIPLE SEEDS

We further repeat the ablation experiments in Sec. 4.3 and Sec. 4.4 across multiple random seeds to examine the robustness of the observed trends, as shown in Fig. 8. Across all five seeds, the results remain consistent with those reported in the main text. In particular, autoregressive generation consistently provides stronger initial proposals than diffusion-only and RL-only, while diffusion-based repair consistently improves upon the autoregressive candidates more effectively than random or autoregressive repair. We also observe the same benefits from distance conditioning, error-guided repair masks, and multiple repair rounds across seeds.

As expected for a best-of-budget discovery metric, the best distance achieved in individual runs exhibits some seed-to-seed variation. However, this variation does not change the overall conclusions of the ablation studies: the relative trends between methods remain stable across independent runs. Moreover, across all seeds, the final discovered codes consistently achieve distance greater than 12, indicating that the framework reliably discovers high-quality codes beyond the distance-12 level of the published BB reference rather than relying on a single favorable run.

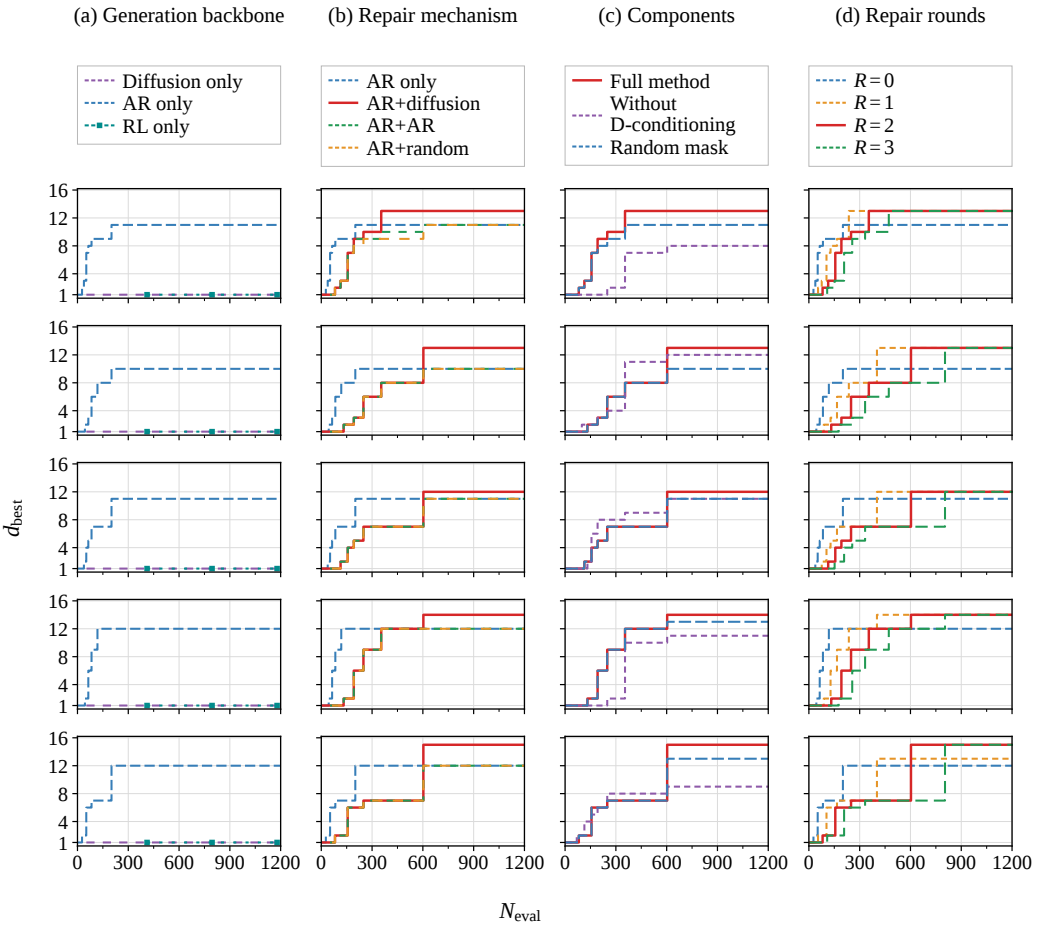



Figure 8: **Results on multiple seeds.**

## G Random Encoding-Circuit Control

We further construct random encoding-circuit controls at $n = 144$ and $k = 15$ to examine whether the performance gain can be obtained without the learned gate-selection distribution.

Specifically, we generate random encoding circuits using the same encoding-circuit representation and the same circuit length as the learned constructions, while replacing the learned gate-selection distribution with random gate choices. We evaluate these random constructions under the same QEC-evaluation budget of $N_{\text{eval}} = 1200$. Despite matching the circuit-length scale of the learned constructions, the best observed code distance remains $d = 1$.

We find that incomplete local coverage is a common failure mode among these random constructions. In particular, a physical qubit that does not participate in any X-type check can admit a weight-one Z logical error, with the analogous failure mode for missing Z-type coverage. We therefore consider a stronger random control by applying a coverage filter before distance evaluation. After constructing each random code, we retain only candidates in which every physical qubit participates in at least one X-type and one Z-type stabilizer check. The random circuits remain matched in length to the learned constructions, and the same $N_{\text{eval}} = 1200$ distance-evaluation budget is used after filtering.

Removing these direct coverage failures increases the best observed distance from $d = 1$ to $d = 3$, but the resulting codes remain substantially below the learned generator.

These results show that random gate selection under the same encoding-circuit representation and circuit-length scale does not reproduce the performance of the learned generator. Even after eliminating simple local coverage failures, the random controls remain far below the discovered codes, indicating that the learned gate-selection distribution captures nontrivial structure relevant to achieving high code distance.

## H Structure of Discovered Codes

The discovered $[[72, 12, 8]]$ and $[[144, 12, 15]]$ codes retain a pronounced sparse-check structure. Their maximum stabilizer check weights correspond to only $\sim 10\%$ of the respective block lengths. Thus, even the heaviest checks act on a relatively small fraction of the physical qubits, placing these constructions well within a sparse-check regime rather than that of generic dense CSS codes.

Within this sparse regime, the learned codes relax the more restrictive check-weight profile of the corresponding BB constructions. This additional structural freedom allows the model to realize larger distances at fixed $(n, k)$, without abandoning sparsity altogether. We therefore view the discovered codes not as dense CSS codes, but as constructions in a broader sparse-check design space surrounding existing QLDPC families.

This interpretation is also consistent with the training distribution, which is dominated by structured QEC families including bivariate-bicycle, generalized-bicycle, hypergraph-product, and lifted-product codes. The generative model consequently inherits a strong prior toward sparse and structured constructions, while retaining the flexibility to depart from the exact locality pattern of any individual family when doing so improves code distance.

From this perspective, the comparison with BB codes is intended to probe the tradeoff between structural rigidity and distance: at fixed $(n, k)$, how much additional distance becomes accessible when the particularly restrictive BB sparsity pattern is modestly relaxed, while remaining far from the dense-code regime? Appendix G further examines this distinction by comparing against random encoding circuits matched in circuit length and representation, testing whether the observed gains can be explained simply by increased circuit density.