KNITNET: LEARNING TO SEARCH AT INFERENCE TIME FOR EXACT RNA INVERSE DESIGN

Anonymous authors
Paper under double-blind review

ABSTRACT

RNA inverse design seeks nucleotide sequences that fold into specified target structures. Existing methods rely on learned generators that can miss exact structural matches, especially beyond their training distributions, or on costly search over a vast, multimodal sequence space. Search is further complicated by globally coupled nucleotide interactions, as correcting one pairing error can create new errors beyond a fixed repair scope. To address this, we introduce KnitNet, an inference-time scaling framework that learns from verifier feedback to refine sequences for exact fold-back matching. Starting from candidates generated for a target secondary structure, KnitNet uses mismatched folds to infer nucleotide dependencies and adapt its joint repair strategy, while an online surrogate learns to predict which edits are promising. This makes search progressively more informed and compute-efficient. Crucially, KnitNet generalizes across RNA structural regimes by changing only the optimization signal to match the verifier required by each setting. Across five secondary-structure benchmarks, KnitNet increases exact solve rates by an average of 21 percentage points over continued sampling with 70% less time-to-solve. On OpenKnot57 specifically, KnitNet is the first method to achieve exact computational matches for all complex, pseudoknotted structures. Beyond 2D, KnitNet improves 3D structural self-consistency over per-target RL by 7.8% while being 39.3% faster, demonstrating generalization and effective inference-time scaling across RNA design regimes.

1 INTRODUCTION

Designing RNA sequences that reliably match a desired structure remains challenging in RNA engineering. Controlling RNA structure could enable realizing desired biological functions, such as programmable regulation of translation or improving the stability and immunogenicity of mRNA vaccines (Green et al., 2014; Zhang et al., 2023). While structure-conditioned sequence generation has become a powerful tool in protein engineering (Dauparas et al., 2022), extending it to RNA three-dimensional (3D) design faces limitations in experimental data and computational validation (Nithin et al., 2024). Secondary structure (2D) offers a complementary, experimentally actionable design layer with substantially greater data coverage (RNAcentral Consortium, 2021; Wayment-Steele et al., 2022; He et al., 2024) and models that support reliable computational fold-back evaluation (He et al., 2024; Lorenz et al., 2011). Together, these properties make secondary structure a biologically meaningful and computationally testable setting for RNA inverse design: *given a target 2D pairing pattern, design a sequence whose predicted fold matches it exactly*.

Given this setting, the central algorithmic challenge is to efficiently find exact solutions in a vast, multimodal sequence space (Figure 1). Supervised RNA generators learn sequence reconstruction given a target structure, but do not optimize exact fold-back correctness (Patil et al., 2024), which could be satisfied by non-native sequences as well. While reinforcement learning can incorporate this folding feedback (He & Sun, 2026), it specializes policies to the training verifier and data distribution, which limits transfer to substantially different design tasks (Panescu et al., 2026)(Figure 1 ①). Inference-time search avoids retraining by adapting candidates directly to each target and verifier. However, repeated folding evaluations are costly, and greedy or narrowly scoped repairs can stall when correcting one interaction disrupts another (Zhou et al., 2023)(Figure 1 ②). Adaptive methods coordinate repairs using structural feedback, but their thermodynamic formulations exclude pseudoknots (Garcia-Martin et al., 2016; Zhou et al., 2026; Kikuchi & Tanaka, 2026), leaving

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.

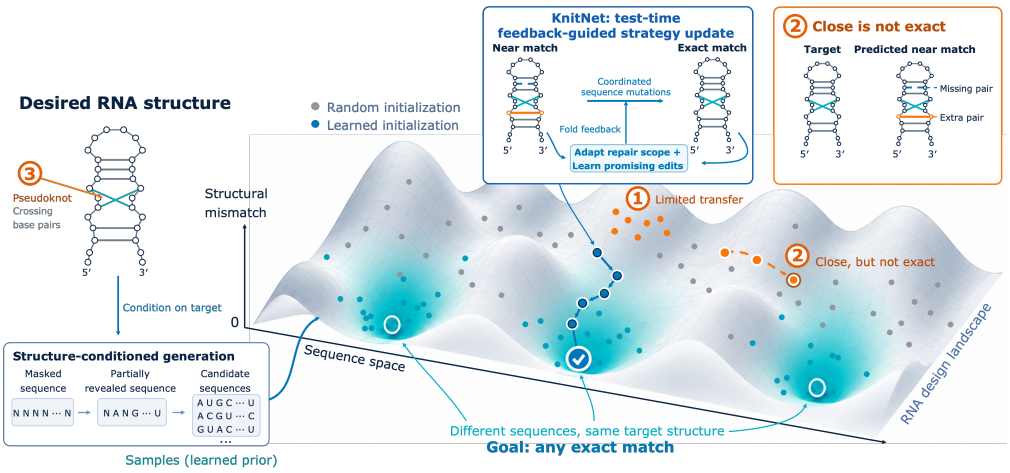

Figure 1: **RNA inverse design in a multimodal sequence landscape.** KnitNet turns test-time compute into exact RNA design solutions through feedback-guided learning and coordinated repair, starting from learned rather than random initializations. Numbered callouts highlight limitations of existing approaches: ① limited transfer across design tasks and folding verifiers; ② search that stalls when repairing one interaction disrupts another; and ③ restrictions to pseudoknot-free structures.

biologically important motifs, such as the CRISPR-Cas12a guide-RNA pseudoknot, outside their scope (Ageely et al., 2021)(Figure 1 ③). Together, these limitations motivate a general RNA design framework that adapts to new targets and verifiers without retraining, uses folding feedback efficiently to resolve coupled structural errors, and supports both planar and pseudoknotted structures.

To this end, we introduce **KnitNet**, a two-stage RNA inverse-design framework that learns from folding feedback at inference time to adapt its search and efficiently find sequences whose predicted structures match the target exactly. **Our main contribution** is *counterexample-guided joint repair* **(1)**, which derives an error graph from observed base-pairing errors to **infer nucleotide interdependencies and expand the editable scope for joint optimization**, overcoming the limitation of local repair. As new errors emerge, KnitNet also revises the repair priorities, while an online surrogate learns which joint mutations are promising and limits expensive folding evaluations to those, making search **compute-efficient**. Crucially, the surrogate is refitted as the repair problem evolves, while also retaining evidence from earlier evaluations. To initialize this repair process, we develop a diffusion generator **(2)** conditioned on a pairing-graph representation of RNA 2D structure, which we construct to support both planar targets and **pseudoknots**. Pretraining on more than 20 million structure-sequence pairs equips the generator with a reusable biological prior, which the repair procedure complements through target-specific inference-time learning. At the framework level, we make this search **transferable** across generators, design regimes, and objectives **(3)** by separating pairing-error-guided repair from verifier-specific candidate ranking.

Across inverse design tasks, we benchmark KnitNet against Order-Agnostic Struct2SeQ (Panescu et al., 2026), the SOTA 2D-conditioned generator, alongside specialized methods for each setting (Appendix A). On OpenKnot57, a challenging pseudoknot benchmark where previous evaluations left targets unsolved despite large sampling budgets (He & Sun, 2026; Townley et al., 2026), KnitNet achieves $57/57$ exact computational matches in individual runs, averaging $56.3/57$ over three seeds versus $49.7/57$ for Order-Agnostic Struct2SeQ. To further demonstrate the value of allocating inference-time computation to feedback-guided learning and repair, we show KnitNet Search achieves higher exact solve rates in 70% less time-to-solve on average for all 20 generator-benchmark combinations in Figure 3 and Appendix L. By applying the same KnitNet procedure to design PDB-derived RNA structures, we improve mean 3D fold-back self-consistency over specialized 3D-conditioned generators, showing that our method can generalize to 3D use-cases, and could help alleviate the data limitations of 3D modeling. What's more, KnitNet search outperforms target-specific RL (Hu et al., 2026), the previously leading 3D method, while being 39.3% faster.

2 THE KNITNET PIPELINE

In this section, we describe KnitNet, an RNA inverse-design framework that combines structure-conditioned sequence generation with adaptive inference-time repair, learning from folding feedback to guide coordinated nucleotide changes so that the predicted fold matches the input structure. For a target of length L, we represent the secondary structure as a pair set $T \subseteq \{(i, j) : 1 \leq i < j \leq L\}$. Each nucleotide belongs to at most one target pair, and crossing pairs are permitted (i.e. pseudoknots). Over the RNA alphabet $\mathcal{A} = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{U}\}$, KnitNet seeks a candidate sequence $x \in \mathcal{A}^L$, with nucleotide x_i at position i. Its predicted pair set $F(x)$ must exactly match the target, $F(x) = T$. Figure 2 summarizes the KnitNet pipeline. First, a frozen structure-conditioned masked diffusion model generates a diverse set of candidate sequences ((Figure 2B ①) (Section 2.1, Appendix D, E). Then, the population search refines these candidates through contextual joint repair: an online local surrogate ranks coordinated nucleotide assignments (Figure 2B ②)(Section 2.2), while verifier-specific scores provide graded search guidance (Figure 2B ③). Exact agreement with the complete target pair set defines success (Section 2.3). Appendix M lists the notation.

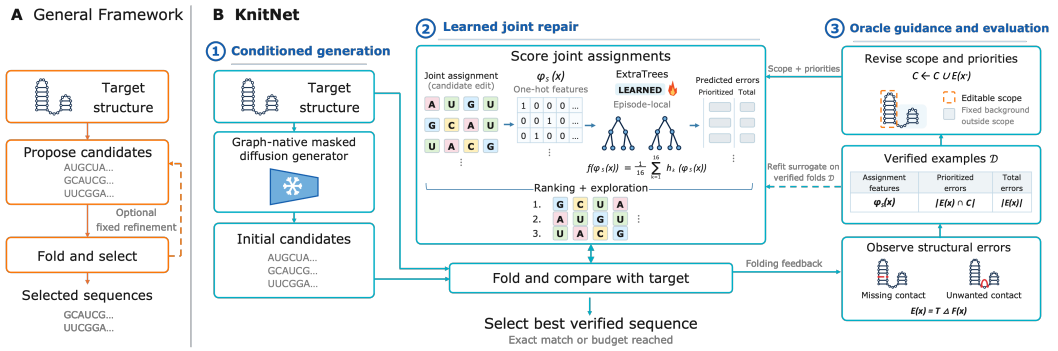

Figure 2: **(A) General RNA inverse design** proposes sequences, evaluates their predicted folds, and selects candidates against a target structure. **(B) KnitNet** combines **(1)** initialization with a frozen structure-conditioned generator, **(2)** coordinated nucleotide edits ranked by a local surrogate, and **(3)** folding feedback that updates the surrogate and revises editable positions and error priorities. This loop adapts subsequent repairs to observed folding errors. Search returns an exact predicted secondary-structure match when found, or the best evaluated sequence when the budget is exhausted.

2.1 GENERATING INITIAL CANDIDATES CONDITIONED ON THE TARGET STRUCTURE

To generate the initial pool of candidate sequences, we use a GraphGPS (Rampášek et al., 2022) denoiser conditioned on a graph representation of the target RNA secondary structure. This enables combining local message passing with global attention to model RNA motif-level interactions alongside long-range dependencies between distant structural regions. We pretrain with a masked diffusion language modeling (MDLM) objective (Sahoo et al., 2024) on a structure-sequence paired dataset we construct by applying the RibonanzaNet-SS 2D structure predictor (He et al., 2024) to the RNACentral sequence corpus (RNAcentral Consortium, 2021). We choose this objective because masked diffusion allows each nucleotide prediction to use revealed sequence context on both sides, helping coordinate assignments across RNA's coupled pairing sites. In contrast, left-to-right autoregressive designers cannot condition on subsequent sequence assignments (Patil et al., 2024; Joshi et al., 2025b). We detail dataset and split construction in Appendix B. Appendix E details graph features and architecture, training objectives and hyperparameters, checkpoint selection, sampling, and training compute.

At inference time, we sample 96 candidates from the learned conditional distribution. For secondary-structure design, we fold each candidate with the selected oracle (RibonanzaNet-SS or ViennaRNA, detailed in Appendix H) and compare its predicted base-pair set $F(x)$ with the target T. If any candidate satisfies $F(x) = T$, the target is recorded as solved; otherwise, the evaluated candidates initialize the search described in Section 2.2. The generator remains frozen during subsequent stages. For 3D inverse design, KnitNet generates the 96-candidate pool conditioned on

the target's DSSR-converted pair graph, and RhoFold+-based tertiary feedback guides subsequent search (Appendix H.2).

2.2 Learning to repair from folding counterexamples

We introduce *counterexample-guided joint repair*, an adaptive inference-time search that learns from folding feedback how to modify nucleotides together to produce sequences that satisfy the coupled pairing constraints of a target structure. This addresses a key challenge in RNA sequence design: interactions between distant nucleotides mean that a local, fixed edit can resolve one pairing error while introducing another elsewhere in the fold. Our contribution couples three adaptive decisions: (1) which interacting nucleotides to optimize jointly, (2) which coordinated edits merit expensive folding calls according to a learned evaluation model, and (3) which errors subsequent repairs should prioritize as new folding feedback becomes available. We call this *counterexample-guided* because errors exposed by candidate folds revise the repair problem used in subsequent iterations. This feedback loop learns and adapts repairs to the target and sequence context at inference time. Algorithm 1 summarizes this procedure.

(1) Constructing a contextual repair problem. We identify a repair problem by folding a complete candidate sequence x and comparing its predicted base-pair set $F(x)$ with the target T. A target pair absent from $F(x)$ is a *missing pair*, and a predicted pair absent from T is an *unwanted pair*. Together, these discrepancies form the error set $E(x) = T \triangle F(x)$. If $E(x) = \varnothing$, the candidate solves the target (line 15). Otherwise, the errors identify what is wrong, but do not themselves reveal which nucleotide changes will correct it. The population $\mathcal{P}$ retains candidate sequences across repair episodes. Each episode selects an anchor $a \in \mathcal{P}$ with unresolved errors $E(a) \neq \varnothing$. The target defines the *assignment-unit set* $\mathcal{V}_T = \{\{i, j\} : (i, j) \in T\} \cup \{\{i\} : 1 \leq i \leq L,\ i \notin \bigcup_{(j,k)\in T}\{j, k\}\}$, consisting of target pairs and unpaired positions. Pair substitutions use the pair-assignment alphabet $\mathcal{A}_{\text{pair}} = \{\text{AU}, \text{UA}, \text{GC}, \text{CG}, \text{GU}, \text{UG}\} \subset \mathcal{A}^2$, read as (x_i, x_j) for the pair (i, j), and unpaired-position substitutions use $\mathcal{A}$. Pair substitutions preserve nucleotide compatibility (Runge et al., 2019; Tang et al., 2026). We check whether the pair forms by folding the modified sequence.

To identify which assignment units to repair together, we construct an *error graph* $\mathcal{G}_a = (\mathcal{V}_T, \mathcal{E}_a)$, whose edges connect the units containing the endpoints of observed folding errors. For example, if a nucleotide pairs with an unintended partner from another target pair, the two assignment units become linked, suggesting that they may require coordinated repair. We include errors from the anchor and its parent lineage to retain evidence of interactions that may disappear and recur during search. The connected component containing the most recurrent current error initializes the editable *scope* $S \subseteq \mathcal{V}_T$. For a singleton component, we add a nearest unit when one is available (Appendix G).

A *joint assignment* z specifies nucleotide values for the units in S, producing a candidate sequence $x \in \mathcal{A}^L$ that matches the anchor a outside those units. Pair values introduced by substitution or random sampling come from $\mathcal{A}_{\text{pair}}$. Beam proposals may retain or inherit incompatible observed pair assignments. Keeping the surrounding sequence unchanged lets us compare the effects of different nucleotide assignments within S. We initialize the prioritized error set $C \subseteq \{(i, j) : 1 \leq i < j \leq L\}$ with anchor errors having at least one endpoint in a unit of S (line 3). Using the oracle merit $m_F(x)$ in Section 2.3, we rank repairs by the lexicographic loss

$$\ell_C(x) = (|E(x) \cap C|,\ |E(x)|,\ -m_F(x)) \in \mathbb{Z}^2_{\geq 0} \times \mathbb{R}. \tag{1}$$

Candidates with fewer errors in C rank first. Ties are broken by fewer total folding errors and then higher oracle merit. This prioritizes the diagnosed conflict while retaining a comparison of the complete folds.

(2) Learning to predict joint repair quality. Evaluating every proposed combination of nucleotide changes would require expensive folding calls. We therefore learn a local surrogate that predicts how many prioritized and total pairing errors would remain after a joint edit, allowing us to rank proposals before selecting candidates for folding evaluation. We implement this with a small ExtraTrees ensemble (Geurts et al., 2006). Successive splits allow each tree to condition its prediction on choices at multiple assignment units, representing how the effect of one change depends on others. This matches the coupled nature of repair: a combination of changes can be useful even when its individual changes are not. A simpler alternative, such as an additive linear surrogate, would lack these interactions between units, which we show boost performance in Table 3.

We fit the surrogate within each repair episode to the current sequence context and repair priorities. The evaluated sequence set $\mathcal{D} \subseteq \mathcal{A}^L$ contains the anchor and all evaluated joint proposals from that episode, with their folding outputs cached. The feature vector $\phi_S(x)$ concatenates one-hot encodings of the assignments in S, with seven entries per pair unit and five per unpaired unit, counting one extra category per unit (Appendix G). Each tree h_k is fitted on $\mathcal{D}$ to predict the prioritized and total error counts, $(|E(x) \cap C|,\ |E(x)|)$. The local surrogate f averages predictions from K trees:

$$f(\phi_S(x)) = \frac{1}{K} \sum_{k=1}^{K} h_k(\phi_S(x)) \in \mathbb{R}^2. \tag{2}$$

We use the two outputs to rank proposals lexicographically (lines 4–5). We use $K = 16$ trees to limit the cost of refitting as the repair scope and error priorities evolve.

To build on promising repairs while preserving alternative combinations, we maintain a proposal beam $\mathcal{H} \subseteq \mathcal{D}$, $|\mathcal{H}| \leq 8$, selected for repair quality and assignment diversity. Its members can seed proposals even when rejected by the outer population $\mathcal{P}$, and all evaluated joint proposals remain available for surrogate fitting (lines 9–10). When the product of the substitution alphabets is small, we enumerate its assignments. Otherwise, we generate candidates through mutation and crossover of beam members and random assignments (line 4). Because surrogate predictions are imperfect, we select proposals using both predicted lexicographic rank and uniform exploration, sampling uniformly until enough observations exist to fit the surrogate (line 5). We also evaluate single-unit mutations from $\mathcal{P}$ to allow simple corrections across the population. These candidates are excluded from $\mathcal{D}$ because they may differ from the anchor outside the units in S. We fold selected candidates to check for exact solutions and record residual errors (lines 6 and 15).

(3) Refining the repair problem through counterexamples. Resolving the prioritized errors may leave other errors unresolved or introduce new ones, whose repair may require changes outside the current scope (Figure 4). These residual errors serve as *counterexamples* to the current repair attempt, guiding updates to both repair priorities and the editable scope. For a nonempty evaluated joint batch $\mathcal{J}$, we choose the best candidate $x^\star \in \arg\min_{x \in \mathcal{J}} \ell_C(x)$ under the lexicographic ranking in Equation 1 (line 8) and update the prioritized set as $C \leftarrow C \cup E(x^\star)$ (line 9). This promotes errors already counted in the total fold error into the primary repair priority while preserving earlier priorities. Before refitting the surrogate, we relabel all observations in $\mathcal{D}$ under the updated C, using cached folds without additional oracle calls (line 10).

We also expand S when $x^\star$ resolves the previously prioritized errors but still misfolds, when search stalls, or when the proposal pool is exhausted, subject to the episode budget (line 13). We add assignment units whose nucleotides appear most often in residual errors across the beam $\mathcal{H}$, breaking ties by proximity along the sequence to the current scope. Previously evaluated sequences retain the anchor's nucleotides at newly editable positions, so we can re-encode them for the expanded scope and reuse their cached folds. At episode boundaries, we rotate anchors to explore different sequence contexts. Appendix G specifies expansion rules, episode limits, and remaining search policies.

2.3 Oracle guidance and evaluation

Applying the search algorithm described above across different folding oracles requires a common way to translate their predictions into search guidance. We therefore use verifier outputs in two complementary ways: to identify which target constraints remain unsatisfied and to score candidate sequences for search. Pairing discrepancies between a candidate's predicted fold and the target structure guide updates to repair priorities and scope, while an oracle-specific score $m_F : \mathcal{A}^L \to \mathbb{R}$ ranks the population $\mathcal{P}$ and breaks ties between repairs with equal prioritized and total error counts. The score $m_F(x)$ depends on the design setting and its associated verifier. Section 3 describes these setting-specific choices, with full definitions in Appendix H.

3 Experiments

We demonstrate KnitNet's design performance and computational efficiency across pseudoknotted, planar, and 3D RNA inverse design tasks. KnitNet search improves designs from diverse generators, while ablations establish the complementary benefits of learned initialization and search. Finally,

search converts additional inference time into exact solutions more effectively than continued sampling. We analyze how learned, counterexample-guided joint repair contributes to this gain.

RNA inverse design tasks and datasets. To assess KnitNet's general applicability, we evaluate three settings that challenge different aspects of RNA design. *(1) Pseudoknotted design* requires resolving complex crossing interactions in functional RNA motifs. We test this capability on OpenKnot57, where previous methods left targets unsolved (Townley et al., 2026), and the larger PseudoBase++ benchmark (Taufer et al., 2009). *(2) Planar design* tests transfer to the established thermodynamic setting, where specialized search methods tackle challenging human-designed puzzles. We evaluate Eterna100-v2 from the Eterna community challenge (Anderson-Lee et al., 2016), alongside Rfam-Learn-Test and Rfam-Taneda to cover naturally derived structures (Runge et al., 2019; Taneda, 2011). *(3) 3D design* tests whether our framework, developed for the 2D regime, can transfer to 3D geometry restraints. We use 64 RNA monomers from the gRNAde Das test split (Joshi et al., 2025a), matching the fold-back verifier's single-chain scope (Appendix C). We ensure that our diffusion model's training corpus has no sequence-homologous overlap with any evaluation benchmark (Appendix B). We further select 500 challenging held-out structures from this corpus, including pseudoknots and RNAs longer than 500 nucleotides (Appendix C), for component ablations.

Experimental setup. To separate generation quality from search effectiveness, we first evaluate KnitNet and off-the-shelf generators using best-of-96 candidates per target, alongside existing search methods with their native initialization and folding-oracle guidance. We then test both stages of KnitNet independently: applying our search to other generators' candidates, as well as initializing existing search methods with KnitNet candidates. Post-initialization budgets are 30 minutes per target for pseudoknotted design, 60 minutes for planar design and ablations, and up to 2,048 logical evaluations for 3D design, in addition to the initial 96 evaluations. To guide search under different folding objectives, we use RibonanzaNet-SS pairing margins, ViennaRNA target ensemble probabilities, and a RhoFold+ GDT/RMSD reward for pseudoknotted, planar, and 3D design, respectively (Appendix H). In 3D search, secondary-structure errors continue to guide repair, while the tertiary reward guides candidate ranking. The main secondary-structure tables report full-benchmark exact coverage and normalized wall-time solve AUC, measuring final recovery and how early solutions are found, as means and sample standard deviations over three seeds. Additional metrics, including Jaccard, MCC, and capped time to solve, appear in Appendix Tables 9 and 10. For 3D design, within each seed we report scTM, $\mathrm{GDT}_{\mathrm{TS}}$, and RMSD for the highest-scTM candidate in each target's final 96-candidate pool. Table values are the mean and sample standard deviation of the 64-target averages over the same three seeds. Appendix H.2 details conditioning inputs, scoring, and search/RL budget accounting.

Baselines. Because existing RNA design methods specialize in different structural inputs and folding objectives, we select generator and search baselines appropriate to each setting, for example aRNAque for pseudoknotted design (Merleau & Smerlak, 2022), RIFT-VAE for planar design (Watanabe et al., 2026), and tertiary-conditioned generators alongside RIDER's target-specific reinforcement learning for 3D design (Hu et al., 2026). Order-Agnostic Struct2SeQ (Panescu et al., 2026), a recent SOTA method, serves as a shared comparator across all three settings. Appendix A provides background on the complete baseline catalogue. Appendices J and K show implementation details.

3.1 Results

KnitNet increases exact solve rates on pseudoknotted RNA targets. KnitNet achieves $57/57$ exact computational fold-back matches on OpenKnot57 in individual runs and is the first method to find exact solutions for all targets in this benchmark. Over three seeds, KnitNet averages $56.3/57$ on OpenKnot57 and $335.7/339$ on PseudoBase++-339, outperforming generator and search baselines on both, with higher solve AUC than other search methods reflecting faster achievement of exact solutions (Table 1). When modularly applying KnitNet Search to every off-the-shelf generator, we observe consistent improvements, averaging 30.2 and 108.3 additional exact solves, respectively, demonstrating usefulness beyond KnitNet's own initialization. Conversely, when initializing existing search methods with KnitNet Generator starting sequences, we again see improvements averaging 9.8 and 67.7 additional exact solves, respectively, supporting the value of sampling from a learned prior informed by biological sequence-structure relationships. We further show that KnitNet maintains its performance lead even when we adapt existing search methods to our own RibonanzaNet-SS derived structural margin for search guidance in Appendix J.

Table 1: **Inverse-design performance on two pseudoknot benchmarks.** We evaluate each generator off-the-shelf and with KnitNet Search. For search methods, we compare their original implementation with initialization from KnitNet Generator starting candidates. Δ denotes gain in solved-target count when applying a KnitNet component (search or initialization), relative to the original baseline. Higher Solve AUC indicates both more and earlier solutions. Values are mean $\pm$ sample SD over three seeds. **Bold** and underlined indicate the best and second-best displayed means, respectively.

Method	Configuration	OpenKnot57			PseudoBase++-339		
		Exact (/57) ↑	Δ	Solve AUC ↑	Exact (/339) ↑	Δ	Solve AUC ↑
A. Off-the-shelf generators							
Struct2Seq	Generation only	30.0 ± 0.00		–	265.0 ± 3.61		–
	+ KnitNet search	56.0 ± 0.00	$+26.0$	0.9583 ± 0.0040	$\mathbf{335.7 \pm 0.58}$	$+70.7$	$\underline{0.9882 \pm 0.0019}$
Struct2Seq OA	Generation only	$\overline{40.3 \pm 1.53}$		–	282.7 ± 4.16		–
	+ KnitNet search	$\mathbf{56.3 \pm 0.58}$	$+16.0$	$\mathbf{0.9754 \pm 0.0032}$	$\underline{335.3 \pm 0.58}$	$+52.6$	0.9879 ± 0.0013
gRNAde 2D	Generation only	8.7 ± 2.52		–	$\overline{175.0 \pm 1.73}$		–
	+ KnitNet search	52.7 ± 1.15	$+44.0$	0.8740 ± 0.0221	331.3 ± 1.53	$+156.3$	0.9649 ± 0.0038
RNAinformer	Generation only	9.7 ± 3.51		–	178.7 ± 5.03		–
	+ KnitNet search	44.3 ± 0.58	$+34.6$	0.7608 ± 0.0064	332.3 ± 2.89	$+153.6$	0.9676 ± 0.0085
B. Existing search methods							
DesiRNA	Original init.	40.7 ± 1.15		0.6869 ± 0.0223	288.3 ± 4.04		0.8300 ± 0.0119
	KnitNet init.	41.0 ± 2.65	$+0.3$	0.6824 ± 0.0436	300.3 ± 2.89	$+12.0$	0.8719 ± 0.0045
aRNAque	Original init.	23.0 ± 2.65		0.2748 ± 0.0448	185.3 ± 4.04		0.4743 ± 0.0059
	KnitNet init.	36.0 ± 2.00	$+13.0$	0.6005 ± 0.0278	301.7 ± 6.03	$+116.3$	0.8685 ± 0.0150
Enzymer	Original init.	5.0 ± 1.73		0.0552 ± 0.0135	204.7 ± 3.51		0.5376 ± 0.0122
	KnitNet init.	21.0 ± 1.00	$+16.0$	0.3605 ± 0.0211	279.3 ± 0.58	$+74.7$	0.8162 ± 0.0026
KnitNet (ours)	Generation only	31.3 ± 1.53		–	273.7 ± 5.03		–
	+ KnitNet search	$\mathbf{56.3 \pm 1.15}$	$+25.0$	$\underline{0.9677 \pm 0.0191}$	$\mathbf{335.7 \pm 0.58}$	$+62.0$	$\mathbf{0.9885 \pm 0.0021}$

Table 2: **Inverse-design performance on three planar benchmarks.** Configurations and reporting conventions follow Table 1.

Method	Configuration	Eterna100			Rfam-Learn-Test			Rfam-Taneda		
		Exact (/100) ↑	Δ	Solve AUC ↑	Exact (/100) ↑	Δ	Solve AUC ↑	Exact (/29) ↑	Δ	Solve AUC ↑
A. Off-the-shelf generators										
Struct2Seq	Generation only	33.0 ± 1.00		–	18.0 ± 3.61		–	8.7 ± 1.15		–
	+ KnitNet search	81.0 ± 0.00	$+48.0$	0.7620 ± 0.0047	$\mathbf{100.0 \pm 0.00}$	$+82.0$	0.9803 ± 0.0008	23.0 ± 0.00	$+14.3$	0.7827 ± 0.0011
Struct2Seq OA	Generation only	40.0 ± 1.73		–	65.0 ± 2.65		–	17.0 ± 1.00		–
	+ KnitNet search	82.0 ± 1.00	$+42.0$	0.7781 ± 0.0042	$\mathbf{100.0 \pm 0.00}$	$+35.0$	0.9966 ± 0.0006	23.7 ± 0.58	$+6.7$	0.7982 ± 0.0176
RIFT-VAE	Generation only	38.0 ± 0.00		–	94.0 ± 0.00		–	17.0 ± 0.00		–
	+ KnitNet search	85.7 ± 1.53	$+47.7$	0.8061 ± 0.0076	$\mathbf{100.0 \pm 0.00}$	$+6.0$	$\mathbf{0.9997 \pm 0.0000}$	$\underline{24.3 \pm 0.58}$	$+7.3$	0.7982 ± 0.0091
B. Existing search methods										
DesiRNA	Original init.	87.7 ± 2.52		0.8461 ± 0.0184	$\mathbf{100.0 \pm 0.00}$		0.9827 ± 0.0015	24.0 ± 0.00		0.8243 ± 0.0007
	KnitNet init.	80.7 ± 2.08	-7.0	0.7773 ± 0.0160	$\mathbf{100.0 \pm 0.00}$	0.0	0.9707 ± 0.0011	23.3 ± 1.15	-0.7	0.7768 ± 0.0265
FastDesign	Original init.	80.3 ± 1.53		0.7793 ± 0.0147	$\mathbf{100.0 \pm 0.00}$		0.9654 ± 0.0010	24.0 ± 0.00		0.8101 ± 0.0016
	KnitNet init.	75.3 ± 3.21	-5.0	0.7373 ± 0.0333	$\mathbf{100.0 \pm 0.00}$	0.0	0.9722 ± 0.0003	24.0 ± 0.00	0.0	0.8132 ± 0.0006
aRNAque	Original init.	72.0 ± 2.00		0.6970 ± 0.0120	$\underline{99.3 \pm 1.15}$		0.8937 ± 0.0078	23.3 ± 0.58		0.7847 ± 0.0034
	KnitNet init.	74.0 ± 0.00	$+2.0$	0.7184 ± 0.0038	$\mathbf{100.0 \pm 0.00}$	$+0.7$	0.9700 ± 0.0008	23.3 ± 0.58	0.0	0.7705 ± 0.0188
NEMO	Native search	$\underline{89.0 \pm 1.00}$		$\underline{0.8579 \pm 0.0060}$	$\mathbf{100.0 \pm 0.00}$		0.9908 ± 0.0004	$\mathbf{24.7 \pm 0.58}$		$\mathbf{0.8399 \pm 0.0130}$
	KnitNet init.	$\mathbf{89.7 \pm 0.58}$	$+0.7$	$\mathbf{0.8620 \pm 0.0055}$	$\mathbf{100.0 \pm 0.00}$	0.0	$\underline{0.9989 \pm 0.0006}$	24.0 ± 0.00	-0.7	$\underline{0.8264 \pm 0.0006}$
KnitNet (ours)	Generation only	36.0 ± 1.73		–	29.7 ± 1.53		–	11.0 ± 1.00		–
	+ KnitNet search	82.0 ± 1.00	$+46.0$	0.7826 ± 0.0024	$\mathbf{100.0 \pm 0.00}$	$+70.3$	0.9759 ± 0.0032	23.7 ± 0.58	$+12.7$	0.7780 ± 0.0228

KnitNet transfers across target distributions and folding objectives. We next apply KnitNet to planar targets, replacing RibonanzaNet-SS structural feedback with ViennaRNA thermodynamic feedback to guide search. Despite this change in objective, KnitNet Search improves exact solve rates for every tested generator on all three benchmarks (Table 2). On Eterna100-v2, it adds 45.9 solves on average, reaching $82.0/100$ from KnitNet Generator candidates and $85.7/100$ from RIFT-VAE candidates. On the naturally derived RNA structures in the two Rfam benchmarks, it consistently reaches $100/100$ on Rfam-Learn-Test and achieves the second-highest mean exact solve rate on Rfam-Taneda at $24.3/29$ with RIFT-VAE candidates. While the leading method in this thermodynamic setting is NEMO (Portela, 2018), its formulation relies on a planar target representation, ViennaRNA-specific scoring, and hand-crafted heuristics. It therefore cannot support pseudoknotted targets and has limited generalization. Similarly, DesiRNA's replica-exchange Monte Carlo exploration appears particularly effective under the thermodynamic objective (Wirecki et al., 2025). However, this advantage does not persist on the pseudoknot benchmarks, where KnitNet achieves higher exact solve rates (Table 1), indicating that DesiRNA's relative strength may be due to setting-specific inductive biases. Taken together, the results across all five secondary-structure benchmarks show that KnitNet does not depend on a single target topology, folding model, reward formulation, data distribution, or initialization distribution. Rather than being specialized to one inverse-folding setting, the same learned repair framework transfers across pseudoknotted and planar targets, structural and thermodynamic verifiers, and synthetic and naturally derived RNAs.

KnitNet Search uses inference-time compute more effectively than continued sampling. Because continued sampling can itself improve solve coverage (Brown et al., 2024), we evaluate inference-time scaling of our verifier-guided repair against drawing more samples from the generator. For every target, generator, and seed, both approaches begin from the same pool of 96 candidates. One applies KnitNet Search, whereas the matched control continues sampling from the corresponding source generator. Both receive the same post-initialization wall-clock budget and stop upon finding an exact solution or reaching the time limit. The KnitNet Search budget includes verifier calls, which dominate its runtime; the counterexample-guided repair operations themselves are comparatively inexpensive. KnitNet Search achieves higher exact coverage in every matched generator-benchmark configuration. On OpenKnot57, it improves exact coverage by 11.7-44.4 percentage points over continued sampling (Figure 3). Across the four remaining secondary-structure benchmarks, KnitNet Search improves final exact coverage by 16.6 percentage points and reduces capped mean time to solve by 69% on average (Appendix L). These results show that allocating additional inference-time compute to our counterexample-guided joint repair method is more effective than allocating it to continued generator sampling.

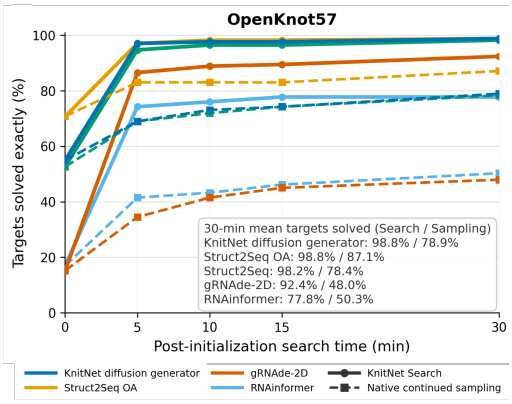

Figure 3: **KnitNet search converts additional inference time into additional exact solutions.** Exact coverage is plotted against post-initialization wall-clock time. Both methods start from identical pools of 96 candidates. Solid curves represent KnitNet search, while dashed curves represent native continued sampling. Curves report means over three seeds.

Ablations. Table 3 shows that the complete KnitNet framework achieves the highest exact coverage (306/500) and AUC (0.5698), with every ablation reducing both metrics. Learned initialization and search provide complementary gains: search from random roots and continued generator sampling reach only 210/500 and 209/500 solves, respectively. Search from random roots appears to have a shorter median TTS (0.17 versus 1.37 minutes). Because this metric includes only targets newly solved during search, its lower value does not imply better overall search efficiency. Instead, this pattern suggests random starts succeed quickly when they land in favorable basins, while learned initialization supports longer refinement that ultimately reaches a broader set of solutions within the budget.

Table 3: **Ablation study of KnitNet framework components.** Each configuration starts with 96 candidates; additional sampling or search runs for up to 60 minutes. Search TTS is median post-initialization time among targets newly solved during search.

Configuration	Exact (/500) ↑	Solve AUC ↑	Search TTS (min) ↓	Jaccard ↑
Full KnitNet search	**306**	**0.5698**	1.37	0.9230
Sampling without search				
Best-of-96	139	0.2780	–	0.8102
Additional sampling	209	0.3954	–	0.8933
Initialization control				
Random init. then search	210	0.4067	**0.17**	0.7365
Search component ablations				
W/ random repair scope	293	0.5435	0.93	0.9133
W/o surrogate ranking	297	0.5515	0.99	0.9178
W/ fixed repair problem	301	0.5598	1.01	0.9164
W/o protected proposal beam	301	0.5600	1.03	0.9184
W/ linear surrogate ranking	301	0.5637	0.87	**0.9265**

Mechanistic analysis shows how failed folds guide KnitNet's adaptive repair. We investigate how learning and refining the repair strategy through verifier feedback enables KnitNet to solve more complex targets than other methods. The search trajectory from a solved PseudoBase++-339 target (Figure 4A,B) illustrates the importance of this mechanism: correcting the initially targeted errors can leave other errors unresolved and introduce new ones, highlighting that a locally successful repair does not guarantee a globally correct fold. Figure 4C (top) shows how KnitNet remembers observed errors and progressively expands the editable scope to include additional nucleotides, allowing potentially correlated structural regions to be revised together. Importantly, the trajectory shows a temporary increase in error count before reaching an exact fold (Figure 4C, bottom), suggesting that successful repair need not improve monotonically. More broadly, repairs that clear prioritized errors but leave errors outside the editable scope occur in 83.1% of OpenKnot57 and 80.2% of Eterna100 search trajectories on targets unsolved at initialization, underscoring the importance of KnitNet's feedback-driven adaptation of the repair scope across distinct target regimes.

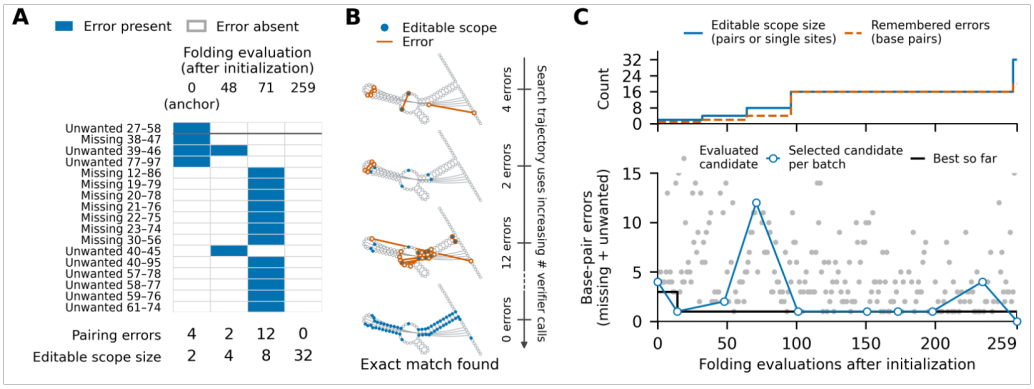

Figure 4: **KnitNet updates joint repair using folding feedback.** **(A)** Evolution of pairing errors during a search trajectory, with snapshots labeled by the number of folding evaluations after initialization. **(B)** Structural visualization of the repair evolution. **(C)** Repair scope and error memory (top) and full-fold error count (bottom). The blue connecting line links successive selected assignments.

KnitNet generalizes to 3D inverse design. Finally, we test whether our learned repair strategy extends from secondary-structure matching to 3D inverse design. Guided by tertiary-structure similarity, KnitNet Search increases mean scTM and GDT_{TS} and reduces RMSD for all six generators across three seeds (Table 4). These gains across 2D- and 3D-conditioned generators support transfer to tertiary-structure objectives, with our method improving even the strongest baseline, RiboDiffusion, from 0.679 to 0.773 mean scTM. We also compare KnitNet Search with RIDER's target-specific RL, the previous leading post-training approach for 3D inverse design (Hu et al., 2026). Starting from the same RIDE initial candidates, KnitNet Search achieves higher mean scTM (0.732 versus 0.679) and lower mean RMSD (2.233 versus 2.691 Å) under the budgets detailed in Appendix H.2. Beyond improving structural self-consistency, KnitNet Search reduces runtime by 39.3% relative to RIDER RL in a separate timing comparison with a matched oracle budget and identical hardware allocation. Together, these results support KnitNet search as a transferable inference-time refinement procedure that improves 3D structural self-consistency.

Table 4: **3D inverse-design performance.** Values are mean ± sample standard deviation of 64 3D targets averages over three seeds.

Method	scTM ↑	GDT_{TS} ↑	RMSD (Å) ↓
A. 2D generators			
KnitNet	0.463 ± 0.005	0.388 ± 0.005	8.864 ± 0.123
+ KnitNet search	0.660 ± 0.005	0.660 ± 0.008	2.964 ± 0.049
Struct2SeQ-OA	0.465 ± 0.006	0.404 ± 0.011	7.555 ± 0.372
+ KnitNet search	0.644 ± 0.008	0.645 ± 0.007	3.028 ± 0.049
B. 3D generators			
gRNAde	0.609 ± 0.003	0.577 ± 0.002	4.651 ± 0.210
+ KnitNet search	0.746 ± 0.005	0.752 ± 0.008	2.218 ± 0.101
NA-MPNN	0.663 ± 0.001	0.646 ± 0.003	3.778 ± 0.103
+ KnitNet search	$\underline{0.771 \pm 0.006}$	$\underline{0.780 \pm 0.007}$	$\underline{1.977 \pm 0.063}$
RIDE	0.553 ± 0.001	0.531 ± 0.001	5.344 ± 0.113
+ KnitNet search	0.732 ± 0.005	0.739 ± 0.005	2.233 ± 0.051
+ RIDER RL	0.679 ± 0.002	0.687 ± 0.002	2.691 ± 0.046
RiboDiffusion	0.679 ± 0.002	0.662 ± 0.003	3.404 ± 0.085
+ KnitNet search	$\mathbf{0.773 \pm 0.004}$	$\mathbf{0.793 \pm 0.001}$	$\mathbf{1.909 \pm 0.133}$

4 Conclusion

We introduced KnitNet, an RNA inverse-design framework that combines a learned, structure-conditioned sequence prior with counter-example-guided joint repair at inference time. KnitNet converts additional test-time computation into exact secondary-structure matches more efficiently than continued sampling across generators, planar and pseudoknotted targets, and folding objectives. Its transfer to 3D design supports the broader applicability of our method. Mechanistic analysis illustrates how verifier feedback is used to expand the repair scope and address globally-coupled nucleotide errors, underpinning KnitNet's distinctive advantage in adapting repair across RNA design regimes. These findings motivate greater emphasis on inference-time scaling through adaptive search, building on learned generators that provide effective starting candidates. Beyond structural objectives, it would be valuable to extend KnitNet to other RNA properties, such as RNA–protein interactions, mRNA stability, and translation. We discuss these future opportunities in Appendix I.

AI USE STATEMENT

We used generative AI tools to aid and polish manuscript writing, retrieve and discover related work, support research ideation and execution, and draft manuscript sections. We did not use these tools to generate synthetic datasets, prove mathematical claims, or for purposes beyond those described above. Separately, we used RibonanzaNet-SS to generate predicted secondary-structure labels for existing RNAcentral sequences, as described in Appendix B. We reviewed all AI-assisted content to ensure accuracy, preserve the intended meaning, and faithfully represent the methods and findings. We take responsibility for the final content of this work, including all text, claims, and artifacts.

REPRODUCIBILITY STATEMENT

We release the pretraining, generation, search, and evaluation code, the pretrained KnitNet generator weights, and the evaluation datasets at `https://anonymous.4open.science/r/knitnet-9094/`. The release includes the five external secondary-structure benchmarks, the internally constructed 500-target ablation panel with its frozen starting candidates, and the 64 reference structures used for 3D evaluation. The repository gives step-by-step instructions for environment setup, sequence generation, and the secondary-structure, ablation, and 3D evaluations, and it records the experiment configurations and random seeds. It provides SHA-256 checksums for datasets and model weights, and it describes how to obtain the third-party folding models and structure-analysis tools from their original distributions. We tested these instructions in a fresh environment. The appendices detail the model architecture, training procedure, evaluation protocols, and dataset construction and overlap checks. The full pretraining corpus is not included in this initial release. We describe its construction, filtering, deduplication, and split procedures in the appendix and intend to release it with the final version of the paper.

REFERENCES

Eman A. Ageely, Ramadevi Chilamkurthy, Sunit Jana, Leonora Abdullahu, Daniel O'Reilly, Philip J. Jensik, Masad J. Damha, and Keith T. Gagnon. Gene editing with CRISPR-Cas12a guides possessing ribose-modified pseudoknot handles. *Nature Communications*, 12 (1):6591, 2021. doi: 10.1038/s41467-021-26989-z. URL https://doi.org/10.1038/s41467-021-26989-z.

Jeff Anderson-Lee, Eli Fisker, Vineet Kosaraju, Michelle Wu, Justin Kong, Jeehyung Lee, Minjae Lee, Mathew Zada, Adrien Treuille, and Rhiju Das. Principles for Predicting RNA Secondary Structure Design Difficulty. *Journal of Molecular Biology*, 428(5):748–757, 2016. doi: 10.1016/j.jmb.2015.11.013. URL https://doi.org/10.1016/j.jmb.2015.11.013.

Mirela Andronescu, Anthony P. Fejes, Frank Hutter, Holger H. Hoos, and Anne Condon. A new algorithm for rna secondary structure design. *Journal of Molecular Biology*, 336(3):607–624, 2004. doi: 10.1016/j.jmb.2003.12.041.

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

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling, 2024. URL https://arxiv.org/abs/2407.21787.

Anke Busch and Rolf Backofen. INFO-RNA: A fast approach to inverse rna folding. *Bioinformatics*, 22(15):1823–1831, 2006. doi: 10.1093/bioinformatics/btl194.

Tristan Cazenave. The montparnasse algorithm for rna design, 2026.

J. Dauparas, I. Anishchenko, N. Bennett, H. Bai, R. J. Ragotte, L. F. Milles, B. I. M. Wicky, A. Courbet, R. J. de Haas, N. Bethel, P. J. Y. Leung, T. F. Huddy, S. Pellock, D. Tischer, F. Chan, B. Koepnick, H. Nguyen, A. Kang, B. Sankaran, A. K. Bera, N. P. King, and D. Baker. Robust deep learning–based protein sequence design using ProteinMPNN. *Science*, 378(6615):49–56, 2022. doi: 10.1126/science.add2187. URL https://doi.org/10.1126/science.add2187.

Andrew Favor, Riley Quijano, Elizaveta Chernova, Andrew Kubaney, Connor Weidle, Morgan A. Esler, Lilian McHugh, Ann Carr, Yang Hsia, David Juergens, Kenneth D. Carr, Paul T. Kim, Yuliya Politanska, Enisha Sehgal, Paul S. Kwon, Robert J. Pecoraro, Cameron Glasscock, Andrew J. Borst, Frank DiMaio, Barry L. Stoddard, and David Baker. De novo design of RNA and nucleoprotein complexes. *bioRxiv*, 2025. doi: 10.1101/2025.10.01.679929. URL https://www.biorxiv.org/content/10.1101/2025.10.01.679929v1.

Juan Antonio Garcia-Martin, Peter Clote, and Ivan Dotu. RNAiFOLD: a constraint programming algorithm for RNA inverse folding and molecular design. *Journal of Bioinformatics and Computational Biology*, 11(02):1350001, 2013. doi: 10.1142/s0219720013500017. URL https://doi.org/10.1142/s0219720013500017.

Juan Antonio Garcia-Martin, Ivan Dotu, Javier Fernandez-Chamorro, Gloria Lozano, Jorge Ramajo, Encarnacion Martinez-Salas, and Peter Clote. RNAiFold2T: Constraint Programming design of thermo-IRES switches. *Bioinformatics*, 32(12):i360–i368, 2016. doi: 10.1093/bioinformatics/btw265. URL https://doi.org/10.1093/bioinformatics/btw265.

Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63:3–42, 2006. doi: 10.1007/s10994-006-6226-1. URL https://doi.org/10.1007/s10994-006-6226-1.

Alexander A. Green, Pamela A. Silver, James J. Collins, and Peng Yin. Toehold Switches: De-Novo-Designed Regulators of Gene Expression. *Cell*, 159(4):925–939, 2014. doi: 10.1016/j.cell.2014.10.002. URL https://doi.org/10.1016/j.cell.2014.10.002.

Stefan Hammer, Birgit Tschiatschek, Christoph Flamm, Ivo L. Hofacker, and Sven Findeiß. RNAblueprint: flexible multiple target nucleic acid sequence design. *Bioinformatics*, 33(18):2850–2858, 2017. doi: 10.1093/bioinformatics/btx263.

Shujun He and Qing Sun. Struct2SeQ: RNA inverse folding with Deep Q-Learning. bioRxiv, 2026. URL https://doi.org/10.64898/2026.01.16.700031. Preprint.

Shujun He, Rui Huang, Jill Townley, Rachael C. Kretsch, Thomas G. Karagianes, David B.T. Cox, Hamish Blair, Dmitry Penzar, Valeriy Vyaltsev, Elizaveta Aristova, Arsenii Zinkevich, Artemy Bakulin, Hoyeol Sohn, Daniel Krstevski, Takaaki Fukui, Fumiya Tatematsu, Yusuke Uchida, Donghoon Jang, Jun Seong Lee, Roger Shieh, Tom Ma, Eduard Martynov, Maxim V. Shugaev, Habib S.T. Bukhari, Kazuki Fujikawa, Kazuki Onodera, Christof Henkel, Shlomo Ron, Jonathan Romano, John J. Nicol, Grace P. Nye, Yuan Wu, Christian Choe, Walter Reade, Eterna participants, and Rhiju Das. Ribonanza: deep learning of RNA structure through dual crowdsourcing. bioRxiv, 2024. URL https://doi.org/10.1101/2024.02.24.581671. Preprint.

Tianmeng Hu, Yongzheng Cui, Biao Luo, and Ke Li. RIDER: 3D RNA Inverse Design with Reinforcement Learning-Guided Diffusion. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 79321–79346, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/80098914b3b3bad79b80377751a85430-Paper-Conference.pdf.

Han Huang, Ziqian Lin, Dongchen He, Liang Hong, and Yu Li. RiboDiffusion: Tertiary structure-based rna inverse folding with generative diffusion models. *Bioinformatics*, 40(Supplement_1): i347–i356, 2024. doi: 10.1093/bioinformatics/btae259.

Natsuki Iwano, Tatsuo Adachi, Kazuteru Aoki, Yoshikazu Nakamura, and Michiaki Hamada. Generative aptamer discovery using RaptGen. *Nature Computational Science*, 2:378–386, 2022. doi: 10.1038/s43588-022-00249-6.

Chaitanya K. Joshi, Edoardo Gianni, Samantha L. Y. Kwok, Simon V. Mathis, Pietro Liò, and Philipp Holliger. Generative inverse design of RNA structure and function with gRNAde, December 2025a. URL https://www.biorxiv.org/content/10.1101/2025.11.29.691298v1. ISSN: 2692-8205 Pages: 2025.11.29.691298 Section: New Results.

Chaitanya K. Joshi, Arian R. Jamasb, Ramon Viñas, Charles Harris, Simon V. Mathis, Alex Morehead, Rishabh Anand, and Pietro Liò. gRNAde: Geometric Deep Learning for 3D RNA inverse design. In *International Conference on Learning Representations*, 2025b. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/1b96f01343ff10150e6719eb163e1536-Paper-Conference.pdf.

Shuta Kikuchi and Shu Tanaka. Factorization machine with quadratic-optimization annealing for RNA inverse folding and evaluation of binary-integer encoding and nucleotide assignment. *Scientific Reports*, 16(1):20460, 2026. doi: 10.1038/s41598-026-50891-7. URL https://doi.org/10.1038/s41598-026-50891-7.

Robert Kleinkauf, Martin Mann, and Rolf Backofen. antaRNA: ant colony-based RNA sequence design. *Bioinformatics*, 31(19):3114–3121, 2015. doi: 10.1093/bioinformatics/btv319. URL https://doi.org/10.1093/bioinformatics/btv319.

Rohan V. Koodli, Benjamin Keep, Katherine R. Coppess, Fernando Portela, Eterna participants, and Rhiju Das. EternaBrain: Automated rna design through move sets and strategies from an internet-scale rna videogame. *PLoS Computational Biology*, 15(6):e1007059, 2019. doi: 10.1371/journal.pcbi.1007059.

Ryan K Krueger and Max Ward. JAX-RNAfold: scalable differentiable folding. *Bioinformatics*, 41 (5):btaf203, May 2025. ISSN 1367-4811. doi: 10.1093/bioinformatics/btaf203. URL https://doi.org/10.1093/bioinformatics/btaf203.

Yichong Li, Xiaoyong Pan, Hongbin Shen, and Yang Yang. DRAG: design RNAs as hierarchical graphs with reinforcement learning. *Briefings in Bioinformatics*, 26(2):bbaf106, 2025. doi: 10.1093/bib/bbaf106. URL https://doi.org/10.1093/bib/bbaf106.

Ronny Lorenz, Stephan H Bernhart, Christian Höner zu Siederdissen, Hakim Tafer, Christoph Flamm, Peter F Stadler, and Ivo L Hofacker. ViennaRNA Package 2.0. *Algorithms for Molecular Biology*, 6(1):26, 2011. doi: 10.1186/1748-7188-6-26. URL https://doi.org/10.1186/1748-7188-6-26.

Nuria Lozano-García, Álvaro Rubio-Largo, and José Maria Granado-Criado. A Simple Yet Effective Greedy Evolutionary Strategy for RNA Design. *IEEE Transactions on Evolutionary Computation*, 29(5):1913–1923, 2025. doi: 10.1109/tevc.2024.3461509. URL https://doi.org/10.1109/tevc.2024.3461509.

Xiang-Jun Lu, Harmen J. Bussemaker, and Wilma K. Olson. DSSR: an integrated software tool for dissecting the spatial structure of RNA. *Nucleic Acids Research*, 43(21):e142–e142, December 2015. ISSN 0305-1048. doi: 10.1093/nar/gkv716. URL https://doi.org/10.1093/nar/gkv716.

David H. Mathews, Matthew D. Disney, Jessica L. Childs, Susan J. Schroeder, Michael Zuker, and Douglas H. Turner. Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of RNA secondary structure. *Proceedings of the National Academy of Sciences*, 101(19):7287–7292, 2004. doi: 10.1073/pnas.0401799101. URL https://doi.org/10.1073/pnas.0401799101.

Nono S. C. Merleau and Matteo Smerlak. aRNAque: An evolutionary algorithm for inverse pseudoknotted rna folding inspired by lévy flights. *BMC Bioinformatics*, 23:335, 2022. doi: 10.1186/s12859-022-04866-w.

Chandran Nithin, Sebastian Kmiecik, Roman Błaszczyk, Julita Nowicka, and Irina Tuszyńska. Comparative analysis of RNA 3D structure prediction methods: towards enhanced modeling of RNA–ligand interactions. *Nucleic Acids Research*, 52(13):7465–7486, 2024. doi: 10.1093/nar/gkae541. URL https://doi.org/10.1093/nar/gkae541.

Takumi Otagaki, Junichi Iwakiri, Goro Terai, Kiyoshi Asai, and Kengo Sato. PKProbDesign: RNA inverse folding including pseudoknots by optimizing thermodynamic folding probability. bioRxiv, 2026. URL https://doi.org/10.64898/2026.07.09.736945. Preprint.

Antonia Panescu, Shujun He, Yixuan He, and Rex Ying. Order-Agnostic Decoding for Sample-Efficient RNA Inverse Folding. Workshop on Generative and Agentic AI for Biology, ICML 2026, 2026. URL https://openreview.net/forum?id=3tWscvIMJG. Non-archival workshop paper.

Sharat Patil, Frederic Runge, Jörg Franke, and Frank Hutter. Towards generative rna design with tertiary interactions. In *ICLR 2024 Workshop on Generative and Experimental Perspectives for Biomolecular Design*, 2024. URL https://openreview.net/forum?id=pLzoHOceHN.

Fernando Portela. An unexpectedly effective Monte Carlo technique for the RNA inverse folding problem. bioRxiv, 2018. URL https://doi.org/10.1101/345587. Preprint.

Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a General, Powerful, Scalable Graph Transformer. In *Advances in Neural Information Processing Systems*, volume 35, pp. 14501–14515, 2022. doi: 10.52202/068431-1054. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/5d4834a159f1547b267a05a4e2b7cf5e-Abstract-Conference.html.

RNAcentral Consortium. RNAcentral 2021: secondary structure integration, improved sequence search and new member databases. *Nucleic Acids Research*, 49(D1):D212–D220, 2021. doi: 10.1093/nar/gkaa921. URL https://doi.org/10.1093/nar/gkaa921.

Frederic Runge, Danny Stoll, Stefan Falkner, and Frank Hutter. Learning to design RNA. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ByfyHh05tQ.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Paul J. Sample, Ban Wang, David W. Reid, Vlad Presnyak, Iain J. McFadyen, David R. Morris, and Georg Seelig. Human 5′ UTR design and variant effect prediction from a massively parallel translation assay. *Nature Biotechnology*, 37(7):803–809, 2019. doi: 10.1038/s41587-019-0164-5.

Tao Shen, Zhihang Hu, Siqi Sun, Di Liu, Felix Wong, Jiuming Wang, Jiayang Chen, Yixuan Wang, Liang Hong, Jin Xiao, Liangzhen Zheng, Tejas Krishnamoorthi, Irwin King, Sheng Wang, Peng Yin, James J. Collins, and Yu Li. Accurate RNA 3D structure prediction using a language model-based deep learning approach. *Nature Methods*, 21:2287–2298, 2024. doi: 10.1038/s41592-024-02487-0. URL https://doi.org/10.1038/s41592-024-02487-0.

Jade Shi, Rhiju Das, and Vijay S. Pande. SentRNA: Improving computational RNA design by incorporating a prior of human design strategies, 2018. URL https://arxiv.org/abs/1803.03146. Preprint.

Martin Steinegger and Johannes Söding. MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature Biotechnology*, 35(11):1026–1028, 2017. doi: 10.1038/nbt.3988.

Blake A. Sweeney, David Hoksza, Eric P. Nawrocki, Carlos Eduardo Ribas, Fábio Madeira, Jamie J. Cannone, Robin Gutell, Aparna Maddala, Caeden D. Meade, Loren Dean Williams, Anton S. Petrov, Patricia P. Chan, Todd M. Lowe, Robert D. Finn, and Anton I. Petrov. R2DT is a framework for predicting and visualising RNA secondary structure using templates. *Nature Communications*, 12(1):3494, June 2021. ISSN 2041-1723. doi: 10.1038/s41467-021-23555-5. URL https://doi.org/10.1038/s41467-021-23555-5.

Cheng Tan, Yijie Zhang, Zhangyang Gao, Bozhen Hu, Siyuan Li, Zicheng Liu, and Stan Z. Li. RDesign: Hierarchical data-efficient representation learning for tertiary structure-based rna design. In *The Twelfth International Conference on Learning Representations*, 2024.

Akito Taneda. MODENA: a multi-objective RNA inverse folding. *Advances and Applications in Bioinformatics and Chemistry*, 4:1–12, 2011. doi: 10.2147/AABC.S14335. URL https://doi.org/10.2147/AABC.S14335.

Akito Taneda. Multi-Objective Genetic Algorithm for Pseudoknotted RNA Sequence Design. *Frontiers in Genetics*, 3:36, 2012. doi: 10.3389/fgene.2012.00036. URL https://doi.org/10.3389/fgene.2012.00036.

Wei Yu Tang, Ning Dai, Tianshuo Zhou, David H. Mathews, and Liang Huang. SamplingDesign: RNA design via continuous optimization with coupled variables and Monte-Carlo sampling. *Nature Communications*, 17(1):2950, 2026. doi: 10.1038/s41467-025-67901-3. URL https://doi.org/10.1038/s41467-025-67901-3.

Michela Taufer, Abel Licon, Roberto Araiza, David Mireles, F. H. D. van Batenburg, Alexander P. Gultyaev, and Ming-Ying Leung. PseudoBase++: an extension of PseudoBase for easy searching, formatting and visualization of pseudoknots. *Nucleic Acids Research*, 37(suppl_1):D127–D135, 2009. doi: 10.1093/nar/gkn806.

The RNAcentral Consortium. RNAcentral in 2026: genes and literature integration. *Nucleic Acids Research*, 54(D1):D303–D313, 2026. doi: 10.1093/nar/gkaf1329.

Jill Townley, Wipapat Kladwang, David Baker, Hamish M. Blair, Christian A. Choe, Gina El Nesr, Andrew Favor, Eli Fisker, Daniel B. Haack, Shujun He, Jason Hingey, Po-Ssu Huang, Rui Huang, Chaitanya K. Joshi, Thomas Karagianes, Andrew Kubaney, Pietro Liò, Adamo Mancino, Jonathan Romano, Boris Rudolfs, Nicholas Spellmon, Navtej Toor, Jigyasa Verma, Vivian Wu, Zhiheng Yu, Eterna Participants, and Rhiju Das. De novo design of RNA pseudoknots with deep learning. *Science*, 393(6814):931–937, 2026. doi: 10.1126/science.aeg6829.

Douglas H. Turner and David H. Mathews. NNDB: the nearest neighbor parameter database for predicting stability of nucleic acid secondary structure. *Nucleic Acids Research*, 38(Database issue):D280–D282, 2010. doi: 10.1093/nar/gkp892. URL https://doi.org/10.1093/nar/gkp892.

Jue Wang, Jintong Dong, Lan Yang, Tianhao Li, Jianwei Yin, Jintao Chen, Ying Dong, Jia Li, and Cheng Tan. Unlocking programmable and creative rna sequence design with RDiffusion. *bioRxiv*, 2026. doi: 10.64898/2026.06.13.732023.

Kentaro Watanabe, Manato Akiyama, and Yasubumi Sakakibara. RIFT-VAE: grammar-conditioned pretraining and latent-space optimization for RNA inverse folding. bioRxiv, 2026. URL https://doi.org/10.64898/2026.08.12.744415. Preprint.

Hannah K. Wayment-Steele, Wipapat Kladwang, Alexandra I. Strom, Jeehyung Lee, Adrien Treuille, Alex Becka, Eterna Participants, and Rhiju Das. RNA secondary structure packages evaluated and improved by high-throughput experiments. *Nature Methods*, 19(10):1234–1242, 2022. doi: 10.1038/s41592-022-01605-0. URL https://doi.org/10.1038/s41592-022-01605-0.

Tomasz K. Wirecki, Grzegorz Lach, Nagendar Goud Badepally, S. Naeim Moafinejad, Farhang Jaryani, Gaja Klaudel, Kalina Nec, Eugene F. Baulin, and Janusz M. Bujnicki. DesiRNA: Structure-based design of rna sequences with a replica exchange monte carlo approach. *Nucleic Acids Research*, 53(2), 2025. doi: 10.1093/nar/gkae1306.

Brian R. Wolfe, Nicholas J. Porubsky, Joseph N. Zadeh, Robert M. Dirks, and Niles A. Pierce. Constrained multistate sequence design for nucleic acid reaction pathway engineering. *Journal of the American Chemical Society*, 139(8):3134–3144, 2017. doi: 10.1021/jacs.6b12693.

Xiufeng Yang, Kazuki Yoshizoe, Akito Taneda, and Koji Tsuda. Rna inverse folding using monte carlo tree search. *BMC Bioinformatics*, 18:468, 2017. doi: 10.1186/s12859-017-1882-7.

Kasra Zandi, Gregory Butler, and Nawwaf Kharma. An adaptive defect weighted sampling algorithm to design pseudoknotted rna secondary structures. *Frontiers in Genetics*, 7:129, 2016. doi: 10.3389/fgene.2016.00129.

Chengxin Zhang, Morgan Shine, Anna Marie Pyle, and Yang Zhang. US-align: universal structure alignments of proteins, nucleic acids, and macromolecular complexes. *Nature Methods*, 19 (9):1109–1115, September 2022. ISSN 1548-7105. doi: 10.1038/s41592-022-01585-1. URL https://doi.org/10.1038/s41592-022-01585-1.

He Zhang, Liang Zhang, Ang Lin, Congcong Xu, Ziyu Li, Kaibo Liu, Boxiang Liu, Xiaopin Ma, Fanfan Zhao, Huiling Jiang, Chunxiu Chen, Haifa Shen, Hangwen Li, David H. Mathews, Yujian Zhang, and Liang Huang. Algorithm for optimized mRNA design improves stability and immunogenicity. *Nature*, 621(7978):396–403, 2023. doi: 10.1038/s41586-023-06127-z. URL https://doi.org/10.1038/s41586-023-06127-z.

Zhengyue Zhang, Gaia Dolcetti, Christian Tyrchan, Leonardo De Maria, Giovanni Bussi, and Werngard Czechtizky. Exploring secondary structure predictions for RNA-targeted drug discovery: Power and challenges. *Journal of Chemical Information and Modeling*, 66(7):4215–4231, 2026. doi: 10.1021/acs.jcim.6c00108.

Tianshuo Zhou, Ning Dai, Sizhen Li, Max Ward, David H Mathews, and Liang Huang. RNA design via structure-aware multifrontier ensemble optimization. *Bioinformatics*, 39(Supplement 1):i563–i571, 2023. doi: 10.1093/bioinformatics/btad252. URL https://doi.org/10.1093/bioinformatics/btad252.

Tianshuo Zhou, David H. Mathews, and Liang Huang. Fast and Versatile RNA Design via Motif-level Divide-and-Conquer and Structure-level Rival Search, 2026. URL https://arxiv.org/abs/2603.02283. Preprint.

APPENDIX

A Related Work **17**

B Pretraining Corpus Statistics and Quality Control **19**

C Evaluation Dataset Statistics and Exposure Audit **21**

D Composition and diversity of generated sequences **23**

E Generator Architecture, Pretraining, and Sampling **24**

F Learned Search Algorithm Pseudocode **26**

G Joint repair implementation details **27**

H Folding oracles and evaluation protocol **28**

I Limitations and Future Work **31**

J Full Pseudoknot Benchmark Results **32**

K Full Planar RNA Structure Benchmark Results **34**

L Inference-Time Scaling **36**

M Math Notation **37**

A Related Work

Generative and diffusion models for RNA design. Structure-conditioned generators learn reusable sequence–structure regularities. RNAinformer uses axial attention over nucleotide interaction matrices, supporting pseudoknots (Patil et al., 2024). For tertiary design, RDesign learns hierarchical representations, gRNAde uses geometric graph networks for single- and multi-state design, and RiboDiffusion conditions diffusion on RNA backbones (Tan et al., 2024; Joshi et al., 2025b; Huang et al., 2024); RDiffusion supports multiple conditions, including secondary and tertiary structure (Wang et al., 2026). These models provide useful priors, but sequence reconstruction does not directly optimize fold-back correctness, and higher native-sequence recovery need not imply better structural self-consistency (Joshi et al., 2025b; Hu et al., 2026). KnitNet uses masked discrete diffusion for bidirectional completion of partially observed sequences (Austin et al., 2021; Sahoo et al., 2024), complemented by joint paired-endpoint assignments to accommodate coupled RNA constraints. The generator supplies diverse initial candidates and remains frozen, while counterexample-guided repair uses failed folds to improve subsequent proposals.

Combining learned generation with refinement. SentRNA combines a prior learned from human-designed sequences with adaptive-walk refinement (Shi et al., 2018), while RIFT-VAE combines grammar-conditioned pretraining with latent-space optimization and tests generated sequences as starting points for existing solvers (Watanabe et al., 2026). These approaches place refinement in promising regions of sequence space. But their native refinement uses prescribed mutation moves or a model-specific latent representation, rather than rebuilding joint repair scopes from recurring folding errors. KnitNet builds on their generation–refinement strategy with adaptive repair scopes and error priorities, operating across generators without access to their latent spaces.

Learning design policies from folding feedback. LEARNA learns a construction policy per target; Meta-LEARNA transfers a policy trained across targets, and Meta-LEARNA-Adapt continues updating it on the new task (Runge et al., 2019). EternaBrain combines mutation choices learned from human play with refinement strategies, while DRAG learns mutation policies over hierarchical RNA graphs (Koodli et al., 2019; Li et al., 2025). Struct2SeQ uses deep Q-learning for secondary-structure-conditioned generation, including pseudoknots, and its order-agnostic extension removes the fixed left-to-right decoding order (He & Sun, 2026; Panescu et al., 2026). For 3D design, RIDER fine-tunes RIDE with structural self-consistency rewards (Hu et al., 2026). These methods align design with structural outcomes, but adapting a policy to a different objective or verifier can require further optimization. KnitNet places target-specific learning in its local repair surrogate, retaining feedback-driven adaptation without updating the generator.

Structure-guided exploration of sequence space. RNA-SSD combines structural decomposition with stochastic local search, while INFO-RNA uses energy-guided initialization and neighbor selection (Andronescu et al., 2004; Busch & Backofen, 2006). Enzymer weights mutation locations by ensemble defect and adapts mutation extent (Zandi et al., 2016). Other exploration strategies include multi-objective genetic search (MODENA), greedy evolution (GREED-RNA), and heavy-tailed mutation sizes (aRNAque) (Taneda, 2012; Lozano-García et al., 2025; Merleau & Smerlak, 2022); ant-colony optimization (antaRNA) (Kleinkauf et al., 2015); and tree or nested Monte Carlo search (MCTS-RNA, NEMO) (Yang et al., 2017; Portela, 2018). DesiRNA combines targeted mutations with replica-exchange Monte Carlo (Wirecki et al., 2025). These strategies help escape locally favorable regions. But broader exploration alone does not identify which interacting positions must change together: correcting one interaction can disrupt another. KnitNet's error-derived repair scopes and surrogate ranking address this coordination problem and prioritize joint edits for evaluation.

Optimization objectives and coupled assignments. SAMFEO combines structure-aware mutations with multifrontier search to optimize ensemble objectives (Zhou et al., 2023); SamplingDesign optimizes coupled sequence distributions using Monte Carlo gradient estimates (Tang et al., 2026); and PKProbDesign extends probability-guided design to pseudoknotted structures (Otagaki et al., 2026). These objectives provide graded guidance, but improving them need not yield an exact decoded fold, and thermodynamic scores do not directly transfer to every learned or tertiary verifier. KnitNet separates oracle-specific ranking from pairing-error repair and exact secondary-structure

verification. This supports different ranking objectives while preserving the repair procedure, including secondary-error guidance alongside tertiary ranking in 3D design.

Coordinated repair and adaptation of the search problem. RNAblueprint uses dependency graphs to sample assignments compatible with multiple structures (Hammer et al., 2017), but compatibility alone does not ensure that the prescribed pairs form. RNAiFold provides large-neighborhood constraint solving, while RNAiFold2T adapts which positions remain fixed according to structural agreement (Garcia-Martin et al., 2013; 2016). FastDesign uses misfolded rivals to identify differential positions and accumulate joint assignment constraints from local energy differences (Zhou et al., 2026). These establish direct precedents for adaptive, failure-guided repair. But their thermodynamic formulations do not directly supply equivalent repair machinery under a learned pseudoknot-capable verifier; FastDesign's pruning specifically relies on a pseudoknot-free energy decomposition. KnitNet's counterexample-guided scope and priority revision instead use empirical complete-fold errors, extending coordinated repair beyond this analytic setting. Its surrogate provides empirical rankings, rather than certified thermodynamic exclusions.

Learning to allocate expensive folding evaluations. Montparnasse adapts rollout policies online through generalized nested rollout policy adaptation (Cazenave, 2026), while RNA FMQA learns a factorization-machine surrogate and optimizes it with quadratic-optimization annealing (Kikuchi & Tanaka, 2026). These methods learn from search evaluations, but place adaptation in a rollout policy or a surrogate for a fixed whole-sequence objective. KnitNet couples its local surrogate with revision of the repair scope and prioritized errors. Our method differs primarily in where learning enters the search loop: rather than directly parameterizing mutation actions or replacing search with a learned policy, we retain an explicit structure-aware proposal mechanism and learn which proposals merit expensive folding evaluations. This separates proposal generation from proposal valuation, while uniform exploration and the protected proposal beam preserve alternatives when surrogate predictions are imperfect.

B PRETRAINING CORPUS STATISTICS AND QUALITY CONTROL

The structure-conditioned generator uses an RNAcentral release 26 corpus with 21,619,209 training examples, 10,000 validation examples, and 2,000 held-out test examples. Each sequence is paired with a RibonanzaNet-SS predicted structure, represented as a canonical base-pair set. Design correctness is assessed separately by fold-back verification.

Structural supervision. RNAcentral combines annotations from multiple databases, including R2DT template-based and family-consensus structures (The RNAcentral Consortium, 2026; Sweeney et al., 2021). Native sequences did not always reproduce their annotated targets in our preliminary fold-back checks. Because sequence–structure relationships are degenerate and depend on molecular context, we refold all retained sequences with RibonanzaNet-SS to align supervision with the verifier. We chose RibonanzaNet as the fold-back oracle because it incorporates experimental SHAPE and DMS data training. Its SS variant is fine-tuned on PDB structures and achieves a cluster-averaged base-pair F_1 of 0.890 on its held-out PDB benchmark (He et al., 2024). We use evaluation mode to disable dropout, fold equal-length groups without padding, and keep the resulting labels fixed throughout training.

We retain sequences of 20–512 nt to prioritize reliable supervision. Reduced prediction accuracy on longer RNAs has been reported in secondary-structure benchmarks (Zhang et al., 2026); 512 nt is our operating limit. We plan to extend this range as improved predictors are validated on longer RNAs.

Split construction. After exact sequence deduplication, we form MMseqs2 clusters at 80% identity and 80% coverage of both sequences. Held-out examples come from cluster representatives absent from training and undergo direct cross-split screening at the same thresholds. The test split combines 1,000 targets sampled uniformly over eligible unique pair sets with 1,000 sampled across 30 strata: five length bins, three paired-fraction bins, and pseudoknot presence or absence. Validation uses remaining eligible representatives and covers the same strata. The test set contains 2,000 distinct target pair sets. The validation set contains 10,000 distinct sequences spanning 6,033 target pair sets, with multiple sequence examples for some structures. Both sets are selected using deterministic hash rankings with seed 20260901.

Due to current computational-resource constraints, for the computationally intensive ablations we use a fixed, outcome-blind subset of 500 of the 2,000 test targets. We construct this subset through two deterministic nested halvings, from 2,000 to 1,000 and then from 1,000 to 500 targets, preserving equal contributions from the uniformly sampled and stratified panels while balancing length, paired fraction, and pseudoknot presence.

B.1 SPLIT STATISTICS AND QUALITY CONTROL

We scan every example in all three splits. All sequences contain only A/C/G/U, sequence and structure lengths agree, and all structures pass the strict extended-dot-bracket parser. Internal audits find no cross-split sequence matches at the specified thresholds, exact sequence containments, or shared identifiers, sequences, or target pair sets. Table 5 summarizes the complete corpus.

Table 5: **RNAcentral split statistics.** Length is min / median / max; GC is nucleotide-weighted. Paired fraction is median [IQR]. PK denotes targets with crossing base pairs.

Split	N	Length (nt)	GC	Paired fraction	PK (%)
Train	21,619,209	20 / 255 / 512	0.523	0.506 [0.441, 0.560]	82.94
Val	10,000	20 / 71 / 512	0.444	0.345 [0.144, 0.491]	44.09
Test	2,000	20 / 187.5 / 512	0.444	0.415 [0.286, 0.520]	67.25

Held-out targets have shorter median lengths, lower GC content and pairing density, and fewer pseudoknots than training. All splits span 20–512 nt and retain both planar and pseudoknotted targets. Figure 5 shows these shifts.

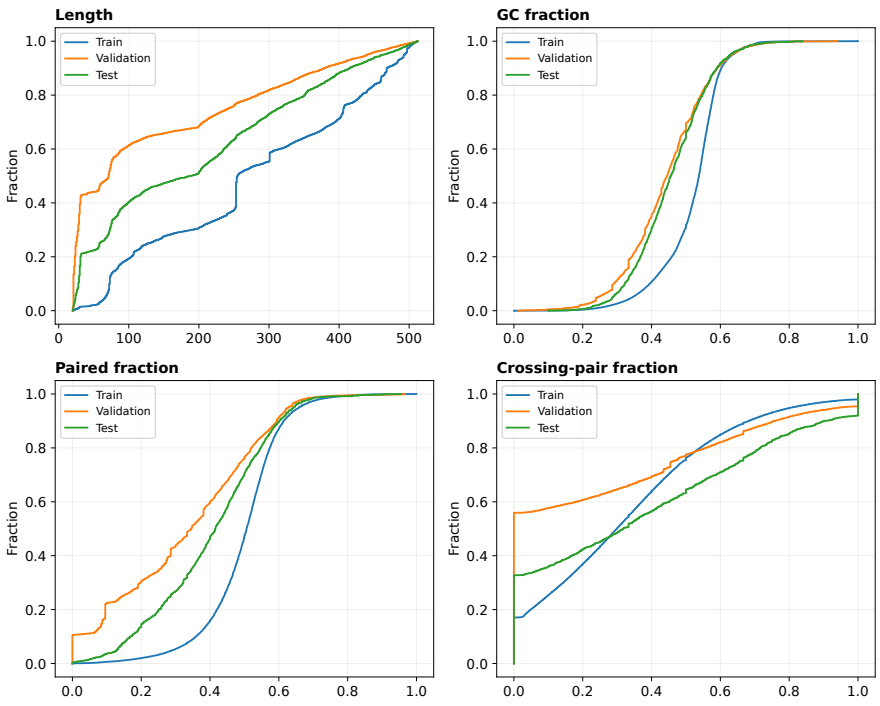

Figure 5: **Training and held-out sequence distributions.**

Structural statistics use actual crossings in the canonical pair set, independent of bracket labels. These checks establish valid, internally separated training pairs; the predicted structures remain pseudolabels.

C Evaluation Dataset Statistics and Exposure Audit

To assess generalization across RNA design tasks, we evaluate on benchmarks spanning different lengths and structural characteristics. Here we establish their complementary coverage and audit their overlap with generator pretraining, providing context for interpreting performance across datasets.

C.1 Benchmark composition and structural diversity

Table 6 summarizes five external secondary-structure benchmarks and the frozen 500-target internal ablation panel. OpenKnot57 and PseudoBase++-339 test pseudoknotted design, whereas Eterna100-v2, Rfam-Learn-Test, and Rfam-Taneda test planar structures. Their median lengths range from 57 to 250 nucleotides, covering both short and long targets. PseudoBase++-339 contains 339 distinct targets after deduplicating 416 valid records by length and nucleotide-pair set. We identify pseudoknots by actual pair crossings, $i < k < j < l$, independently of dot-bracket notation.

Table 6: Benchmark composition and overlap with generator pretraining. Length: min/median/max in nucleotides. PK: percentage with pseudoknots. Exact: targets matching our train set in length and pair set. NN: nearest-training equal-length pair-set Jaccard, p95/max; undefined scores are excluded.

Dataset	N	Verifier	Length	PK (%)	Exact	NN p95 / max
OpenKnot57	57	RibonanzaNet-SS	48 / 100 / 240	100.0	0/57	0.484 / 0.606
PseudoBase++-339	339	RibonanzaNet-SS	21 / 57 / 418	100.0	0/339	0.750 / 0.960
Eterna100-v2	100	ViennaRNA	12 / 104 / 400	0.0	0/100	0.595 / 0.889
Rfam-Learn-Test	100	ViennaRNA	51 / 250 / 447	0.0	0/100	0.550 / 0.892
Rfam-Taneda	29	ViennaRNA	55 / 152 / 452	0.0	0/29	0.740 / 0.955
Ablation-500 (internal)	500	RibonanzaNet-SS	20 / 182 / 511	67.2	0/500	0.893 / 0.986

Figure 6 extends this comparison. The differing distributions show that the panels cover complementary combinations of target size, pairing density, and stem organization, motivating evaluation across all five external benchmarks.

C.2 Overlap with generator pretraining

To evaluate generalization beyond memorized targets, we exclude benchmark matches from generator pretraining and audit the retained corpus. We summarize the checks here; corpus construction is detailed in the following appendix. Structural comparisons use canonical nucleotide-pair sets: an exact match requires equal length and identical pairs. To quantify similarity beyond exact matches, we compute

$$J_{\text{pair}}(T, Q) = \frac{|T \cap Q|}{|T \cup Q|}, \tag{3}$$

for target pair set T, equal-length training pair set Q, and nonempty union $T \cup Q$. Each target's nearest-training score is the maximum over equal-length training candidates. Scores are undefined when no such candidate exists, including three Eterna100-v2 targets below the minimum training length.

None of the five external benchmarks or Ablation-500 contains an exact training target (Table 6). Figure 7 shows the distribution of nearest-training similarities, revealing different degrees of structural proximity despite this absence of duplicates. OpenKnot57 is particularly distinct under this measure: its nearest-training Jaccard never exceeds 0.606, whereas some targets in other panels have close neighbors, but still not exact matches.

We also screen available reference or native sequences, following the use of sequence-similarity filtering in RNA inverse-design benchmarks (Huang et al., 2024). Our audit detects no exact full-sequence matches or MMseqs2 hits (Steinegger & Söding, 2017) meeting both $\geq 80\%$ identity and $\geq 80\%$ coverage of each sequence; exclusions extend to complete sequence clusters containing matched members. These checks establish separation under the stated criteria, not the absence of shared RNA families or motifs. Because multiple sequences can realize the same target structure, sequence checks apply to supplied references rather than all possible design solutions.

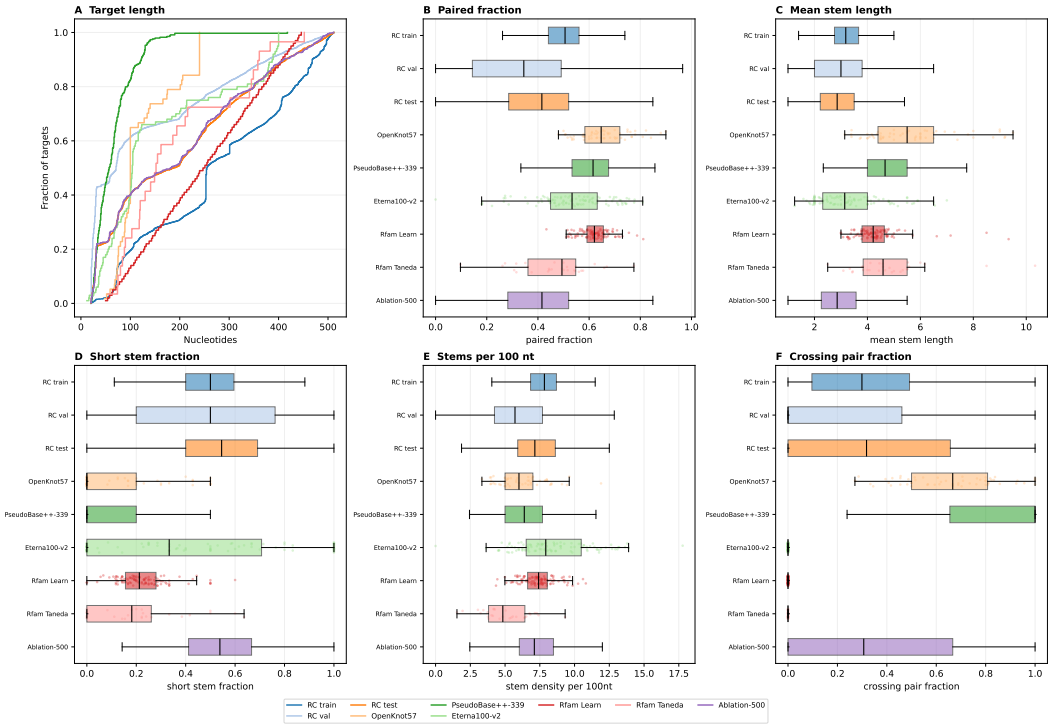

Figure 6: **Structural diversity across evaluation benchmarks.** Target length and five pairing and stem descriptors, with RNAcentral splits as references.

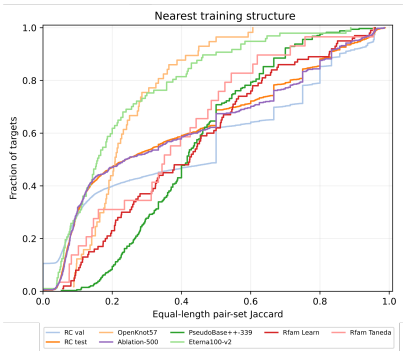

Figure 7: **Structural similarity to generator pretraining.** Cumulative distributions of each target's maximum pair-set Jaccard to an equal-length training structure. Lower Jaccard scores indicate less overlap; targets without an equal-length neighbor are omitted.

C.3 3D Evaluation Set Curation

We use the Das test split released in the gRNAde repository (Joshi et al., 2025b), following its use by RIDER as a common benchmark for 3D RNA inverse design (Hu et al., 2026). We select the first stored conformer for each of its 98 records. We exclude 20 multi-chain records and 14 single-chain records with DSSR-detected backbone breaks, leaving 64 continuous single-chain targets. These criteria are applied before inspecting any model output. DSSR v2.9.3 (Lu et al., 2015), with default annotations, provides the secondary-structure targets, including pseudoknots. We standardize inputs to meet each generator's atom requirements while retaining the original experimental coordinates for reward computation and final evaluation.

D COMPOSITION AND DIVERSITY OF GENERATED SEQUENCES

We characterize the sequences designed by KnitNet, Struct2Seq, and Struct2Seq OA using the first successful design for each target, restricting the comparison to 167 target–seed cases solved by all three methods across three seeds on OpenKnot57 (Figure 8). KnitNet produces more diverse sequences for the same target across seeds, with a median target-level Hamming distance of 57.7%, compared with 37.7% for Struct2Seq and 48.6% for Struct2Seq OA. Its mean GC content (51.8%) is also closest to the native reference (51.6%), compared with 48.1% and 43.0%, respectively, although its GC distribution is wider and other composition differences are mixed. These statistics characterize successful computational designs and suggest the methods output biologically-feasible sequences.

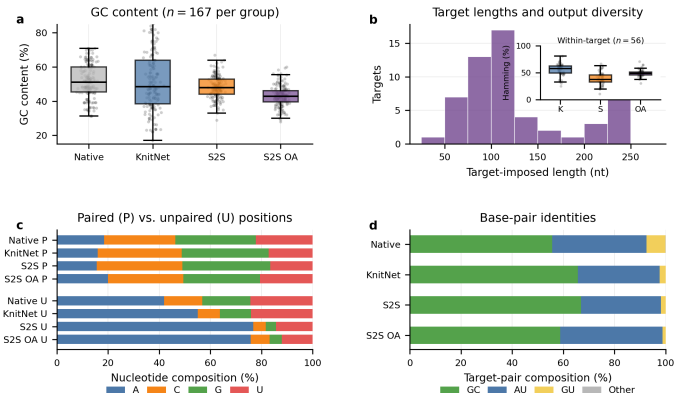

Figure 8: **Composition and diversity of matched OpenKnot57 outputs.**

D.1 COMPOSITION AND DIVERSITY OF 3D GENERATOR OUTPUTS

We characterize all 96 initial sequences generated for each of the 64 three-dimensional targets at seed 42, giving 6,144 sequences per generator and 36,864 generated sequences in total. Every generated sequence is included regardless of its fold-back score. Search and target-specific RL outputs are excluded, so this analysis describes the original generator distributions rather than selected final designs.

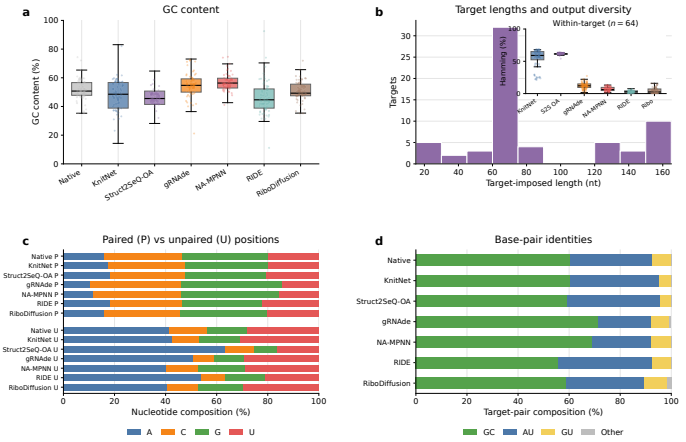

Figure 9: **Composition and diversity of 3D generator outputs.**

KnitNet produces a broad initial sequence distribution, with 54.1% mean within-target Hamming diversity, second to Struct2SeQ-OA at 61.2%. The other four generators yield substantially more concentrated pools, with mean diversity between 3.0% and 12.2%. KnitNet's mean GC content is 48.8%, compared with 51.3% for the native references. The paired-position, unpaired-position, and target-pair compositions in Figure 9 further show that the generators occupy distinct sequence distributions even when conditioned on the same targets.

E GENERATOR ARCHITECTURE, PRETRAINING, AND SAMPLING

E.1 GRAPH REPRESENTATION AND FEATURES

Each nucleotide is a node, with bidirectional backbone and target base-pair edges, including crossing pairs for pseudoknots. Each node has 31 structural features: pairing status, normalized sequence position, eight Laplacian positional encodings, 16 random-walk return probabilities, and five descriptors of loop size, stem size, nesting depth, junction degree, and distance to a sequence end. Each directed edge has 18 features: edge type (2), binned sequence separation (10), bracket layer (1), and indicators of stem, loop, and crossing relationships (5). Structural features depend only on the target structure; nucleotide-or-mask and diffusion-noise embeddings are added separately.

E.2 DENOISER ARCHITECTURE

The denoiser has 12 GraphGPS layers, hidden dimension $d = 512$, eight attention heads, and approximately 36.2 million parameters. Each layer combines edge-aware GATv2 message passing with global self-attention, normalization, and residual connections. For each nucleotide, we sum a projected structural feature vector, a nucleotide-or-mask embedding, and an embedding of diffusion noise level $\sigma(t)$.

Dropout rates are 0.15 within GraphGPS blocks, 0.2 for global attention, and 0.1 after each block and in the pair head.

With generator parameters θ, a linear nucleotide head maps node embeddings $\mathbf{h}_i \in \mathbb{R}^d$ to four logits. Their softmax gives the nucleotide distribution $p^{(4)}_{\theta,i} : \mathcal{A} \to [0,1]$ at position i. For an ordered target pair (i,j) with $i < j$, the pair-head multilayer perceptron $g_\theta : \mathbb{R}^{4d} \to \mathbb{R}^6$ predicts the joint assignment distribution $p^{(6)}_{\theta,ij} : \mathcal{A}_{\text{pair}} \to [0,1]$:

$$p^{(6)}_{\theta,ij} = \text{softmax}(g_\theta([\mathbf{h}_i, \mathbf{h}_j, \mathbf{h}_i \odot \mathbf{h}_j, |\mathbf{h}_i - \mathbf{h}_j|])) \,. \tag{4}$$

The pair head has widths $2048 \to 256 \to 512 \to 6$, using SiLU and dropout after the first linear layer and layer normalization before the final projection. Its classes are the six assignments in $\mathcal{A}_{\text{pair}}$.

E.3 MASKED DIFFUSION OBJECTIVE

For each training sequence, we sample diffusion time t uniformly from $[\epsilon, 1]$ and clip it to $[\epsilon, 1-\epsilon]$, with clipping constant $\epsilon = 10^{-3}$. The log-linear noise schedule is $\sigma(t) = -\log[1-(1-\epsilon)t]$. Nucleotides are independently masked with probability $\mu(t) = 1 - \exp[-\sigma(t)]$ (Sahoo et al., 2024). The time weight is $w(t) = \min\{100, \sigma'(t)/(\exp[\sigma(t)] - 1)\} = \min\{100, 1/t\}$.

For a minibatch $\mathcal{B}$, each example $b \in \mathcal{B}$ has a sequence x_b, a sampled diffusion time t_b, and a corrupted sequence $\widetilde{x}_{b,t_b}$ whose positions hold nucleotides or mask tokens. The masked-position set $\mathcal{M}_\mathcal{B}$ contains indices (b,i). The supervised-pair set $\mathcal{P}_\mathcal{B}$ contains indices (b,i,j) for target pairs with both endpoints masked and observed chemistry in $\mathcal{A}_{\text{pair}}$. Both heads condition on the corrupted sequence, target structure, and diffusion time. We add the example index b to each head's distribution and minimize

$$\begin{aligned}
\mathcal{L}_{\text{nt}} &= -\frac{1}{|\mathcal{M}_\mathcal{B}|} \sum_{(b,i)\in\mathcal{M}_\mathcal{B}} w(t_b) \log p^{(4)}_{\theta,b,i}(x_{b,i}), \\
\mathcal{L}_{\text{pair}} &= -\frac{1}{|\mathcal{P}_\mathcal{B}|} \sum_{(b,i,j)\in\mathcal{P}_\mathcal{B}} w(t_b) \log p^{(6)}_{\theta,b,ij}(x_{b,i}x_{b,j}), \\
\mathcal{L}_{\text{pre}} &= \mathbb{E}[\mathcal{L}_{\text{nt}} + 0.25\mathcal{L}_{\text{pair}}].
\end{aligned} \tag{5}$$

The expectation averages over minibatches, diffusion times, and masks. An empty supervision set contributes zero. Training uses independent nucleotide masking; joint endpoint revelation is used during sampling.

Table 7: Optimization settings for the released generator.

Setting	Value
Optimizer	AdamW, $\beta_1 = 0.9$, $\beta_2 = 0.98$
Weight decay	0.01
Peak learning rate	3×10^{-4}
Warmup	Linear from 10^{-6} for 105,565 updates (10%)
Decay	Cosine, minimum learning rate 10^{-5}
Effective batch size	1,024 sequences
Per-GPU batch / accumulation	256 / 1
Training hardware	$4\times$ NVIDIA H200, distributed data parallel
Numerical precision	Mixed bfloat16
Gradient-norm clipping	1.0
Configured duration	50 epochs / 1,055,650 updates
Training seed	42

E.4 Training recipe and checkpoint selection

The released model was trained on the synthetic RNAcentral corpus containing 21,619,209 training sequences with a 512-nucleotide length cap, with 10,000 validation and 2,000 test examples. Corpus construction and split definitions are described in Appendix B. Table 7 summarizes optimization. We use the checkpoint at update 802,294 (epoch 38) from a run configured for 50 epochs; the learning-rate schedule spans the full configured duration.

Every two epochs, we generate 16 candidates for each target in a fixed 1,000-target validation subset and fold them with RibonanzaNet-SS. We also evaluate the full 10,000-target validation split every ten epochs. Checkpoints are ranked by the trailing two-evaluation mean of best-of-16 pair Jaccard, averaged over the fixed subset. Validation sampling uses a fixed base seed of 1234 across checkpoints. Folding is used for validation, rather than in the training loss.

E.5 Candidate sampling

We generate 96 initial candidates per target. Each trajectory starts fully masked and uses 32 reverse transitions on a uniform time grid from 1 to ϵ. At each transition from t to $s < t$, an unrevealed unit remains masked with probability $\mu(s)/\mu(t)$; otherwise its assignment is sampled. A unit is either an unpaired nucleotide or a target pair, whose endpoints are revealed together using the joint head. Paired assignment follows its established use in RNA design (Runge et al., 2019; Tang et al., 2026). We apply temperature 0.8 and top-$k = 2$ truncation separately to the four-way nucleotide and six-way pair distributions. Revealed assignments remain fixed. An additional denoiser pass fills any remaining masks by argmax, retaining joint pair assignments. Validation uses the same sampler with 16 candidates per target. Evaluation seeds are specified with each benchmark protocol.

E.6 Training compute

Training to the selected checkpoint used four H200 GPUs, totaling 552 H200 GPU-hours.

F LEARNED SEARCH ALGORITHM PSEUDOCODE

Algorithm 1 states the search for secondary-structure design, where a candidate solves the target when $F(x) = T$. The budget B and spent count b measure logical folding evaluations after initialization. The batches $\mathcal{J}$ and $\mathcal{U}$ contain evaluation requests, and their sizes count repeated requests separately. The operation $\mathrm{uniq}(\cdot)$ extracts a batch's set of distinct sequences. The unset-surrogate marker $f = \perp$ triggers uniform joint-proposal sampling. For 3D design, Appendix H.2 keeps these secondary-structure repairs but ranks and exports candidates by a tertiary reward and uses a different stopping rule.

Algorithm 1 Counterexample-guided joint repair for secondary-structure design

Require: Target T, oracle (F, m_F), evaluated candidates $\mathcal{R}$, post-initialization budget B

Cache $\mathcal{R}$; $\mathcal{P} \leftarrow$ SELECTPOPULATION$(\mathcal{R})$; $b \leftarrow 0$; close episode
while $b < B$ and no cached candidate solves T **do**
 # Start a scoped repair episode
  if episode closed, $(a, S, C, \mathcal{D}, \mathcal{H}) \leftarrow$ STARTEPISODE$(\mathcal{P})$
 # Propose and rank joint assignments
  $\mathcal{Q} \leftarrow$ JOINTPROPOSALS$(a, S, \mathcal{H}) \setminus \mathcal{D}$; $f \leftarrow$ FITEXTRATREES$(\mathcal{D}, S, C)$ if $\mathcal{Q} \neq \varnothing$ and $|\mathcal{D}| \geq 8$; else $f \leftarrow \perp$
  $(\mathcal{J}, \mathcal{U}) \leftarrow$ SELECTBATCH$(\mathcal{Q}, f, \mathcal{P}, B - b)$
 # Verify complete folds
  Fold and cache $\mathcal{J}$ and $\mathcal{U}$; $b \leftarrow b + |\mathcal{J}| + |\mathcal{U}|$
 # Refine counterexamples and relabel observations
  if $\mathcal{J} \neq \varnothing$ **then**
    Choose $x^\star \in \arg\min^{\mathrm{lex}}_{x \in \mathcal{J}} \ell_C(x)$; $r \leftarrow$ RECORDPROGRESS$(x^\star, \mathcal{J}, \mathcal{D}, C)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \mathrm{uniq}(\mathcal{J})$; $C \leftarrow C \cup E(x^\star)$
    Relabel $\mathcal{D}$ from cached folds; $\mathcal{H} \leftarrow$ QUALITYDIVERSITYBEAM$(\mathcal{D}, S, C)$
  end if
 # Update the population and expand the repair scope
  $\mathcal{P} \leftarrow$ SELECTPOPULATION$(\mathcal{P} \cup \mathrm{uniq}(\mathcal{J}) \cup \mathrm{uniq}(\mathcal{U}))$; close episode if episode budget exhausted or ($\mathcal{Q} = \varnothing$ and S cannot grow)
  if episode open and $\big(\mathcal{Q} = \varnothing \vee (\mathcal{J} \neq \varnothing \wedge r)\big)$, $S \leftarrow$ EXPANDSCOPE$(S, \mathcal{H})$
end while
return an exact cached candidate, or the best cached candidate marked unsolved

G JOINT REPAIR IMPLEMENTATION DETAILS

Population and episode budget. We retain up to 32 distinct sequences, ranked by oracle merit, then pair Jaccard and fewer pairing errors. Anchors rotate through this population. We cap each episode at 512 joint evaluations to limit search in one sequence context. Each batch contains at most 32 evaluations. For a batch of $n = |\mathcal{J}| + |\mathcal{U}|$ evaluation requests with $\mathrm{uniq}(\mathcal{J}) \subseteq \mathcal{Q}$, we reserve at least $\max(1, \lfloor n/4 \rfloor)$ slots for ordinary single-unit mutations to preserve simple corrections outside the current repair scope. Unused joint slots also go to these mutations. Both types consume the overall search budget.

Scope construction and expansion. In $\mathcal{G}_a$, an error whose endpoints belong to the same assignment unit produces a self-loop. Repeated observations contribute to the error's recurrence count rather than parallel graph edges. We count each anchor error twice and each occurrence in its preceding 15 ancestors once, emphasizing current errors while retaining recurring conflicts. We initialize the scope with the error-graph component containing the highest-count current error. For a singleton component, we add the nearest other assignment unit along the sequence, when available, to permit a coupled edit. We expand after four consecutive joint batches without lexicographic improvement over the best preceding episode observation, comparing both under the current prioritized error set C. We also expand when the previously prioritized errors are resolved but others remain, or when proposals are exhausted. In Algorithm 1, RECORDPROGRESS returns $r = \mathrm{true}$ when one of the first two conditions holds, and exhausted proposals appear as $\mathcal{Q} = \varnothing$. Each expansion adds up to $\max(2, |S|)$ units, prioritizing their endpoint incidence in residual errors across the proposal beam and breaking ties by sequence distance to S. Exhaustion at full scope ends the episode.

Proposal generation. We enumerate the product of the pair-substitution and nucleotide alphabets when it contains at most 4096 assignments. For larger spaces, we combine all single-unit neighbors of beam members, 1024 unit-wise crossover proposals with mutations at up to two units, and 256 random assignments. We remove duplicates and previously evaluated episode sequences before selection; constructing the pool requires no folding calls.

Surrogate and exploration. We fit a multi-output ExtraTrees regressor after eight distinct observations, including the anchor, using 16 trees with maximum depth 10, minimum leaf size one, all features eligible at each split, and random seed 42. One-hot features follow a fixed unit order and include an additional category for incompatible observed pair assignments. For a joint quota $q > 0$, we reserve $\max(1, \lfloor q/4 \rfloor)$ evaluations for uniform exploration and select the remainder by predicted lexicographic rank. Before fitting, selection is entirely uniform. This preserves exploration when predictions are inaccurate.

Protected proposal beam. We retain up to six assignments by repair loss, then add up to two that greedily maximize minimum assignment Hamming distance to those retained. This preserves alternative combinations independently of outer-population selection. Only the best newly evaluated joint proposal contributes errors to the prioritized error set C, preventing arbitrary exploratory failures from expanding it. All joint observations remain available for refitting, with labels recomputed from cached folds without additional evaluations.

H FOLDING ORACLES AND EVALUATION PROTOCOL

H.1 TWO-DIMENSIONAL FOLD-BACK EVALUATION

We use complete predicted pair sets to construct repair problems and determine exactness. Pair discrepancies provide a shared interface across the secondary-structure oracles, while continuous, oracle-specific merits distinguish candidates with similar residual errors. These merits rank candidates within each oracle; their numerical values are not compared across oracles.

OpenKnot57 (RibonanzaNet-SS). RibonanzaNet-SS supplies directed pair scores and a pseudoknot-capable fold (He et al., 2024; He & Sun, 2026). We use Arnie's Hungarian decoder with threshold $\tau = 0.5$ and minimum helix length one. For candidate sequence x, the directed pair scores $q_{ij} \in [0, 1]$ form an $L \times L$ matrix after masking $|i - j| \leq 3$. We suppress their dependence on x and define $b_{ij} = \mathrm{logit}(q_{ij})$, $\bar{b}_{ij} = \mathrm{logit}((q_{ij} + q_{ji})/2)$, and $b_\tau = \mathrm{logit}(\tau)$, clipping logit arguments to $[10^{-7}, 1 - 10^{-7}]$. We define the margins $c_{ij}, c_i : \mathcal{A}^L \to \mathbb{R}$ by

$$\begin{aligned} c_{ij}(x) &= \bar{b}_{ij} - \max\left\{ b_\tau, \max_{k \notin \{i,j\}} b_{ik}, \max_{k \notin \{i,j\}} b_{jk} \right\}, \quad (i,j) \in T, \\ c_i(x) &= b_\tau - \max_{k \neq i} b_{ik}, \qquad\qquad\qquad\qquad\quad i \text{ unpaired in } T, \end{aligned} \tag{6}$$

and use the weakest constraint as the merit:

$$m_F(x) = \min_{c \in \mathcal{C}(T)} c(x). \tag{7}$$

The constraint-margin set $\mathcal{C}(T) = \{c_{ij} : (i,j) \in T\} \cup \{c_i : i \text{ unpaired in } T\}$ contains the margin functions for all target-pair and target-unpaired constraints. Positive margins favor target pairs over the threshold and competing partners, while suppressing pairing at unpaired sites. Maximizing the minimum focuses search on the least satisfied requirement; exactness is checked separately on the decoded fold.

Eterna100 (ViennaRNA). We use ViennaRNA 2.6.4 with Turner 2004 parameters at 37 °C (Anderson-Lee et al., 2016; Lorenz et al., 2011; Mathews et al., 2004; Turner & Mathews, 2010). Following thermodynamic inverse-design objectives (Zhou et al., 2023; Tang et al., 2026), we rank candidates by target ensemble probability:

$$\log P_{\mathrm{V}}(T \mid x) = -\frac{G(x,T) - G_{\mathrm{ens}}(x)}{R\mathcal{T}}, \qquad G_{\mathrm{ens}}(x) = -R\mathcal{T} \log Z(x), \tag{8}$$

Here the target free energy $G(x,T)$ and ensemble free energy $G_{\mathrm{ens}}(x)$ are measured in kcal/mol. The partition function $Z(x)$, gas constant R in kcal mol^{-1} K^{-1}, and absolute temperature $\mathcal{T} = 310.15$ K determine the ensemble free energy. This rewards the target relative to competing folds, rather than lowering its energy alone. The incompatibility count n_{inc} is the number of target pairs assigned nucleotides outside $\mathcal{A}_{\mathrm{pair}}$. We use

$$m_F(x) = \begin{cases} 1 + [1 - \log P_{\mathrm{V}}(T \mid x)]^{-1}, & n_{\mathrm{inc}} = 0, \\ -n_{\mathrm{inc}}/|T|, & n_{\mathrm{inc}} > 0. \end{cases} \tag{9}$$

Since $P_{\mathrm{V}}(T \mid x) \in (0, 1]$, the denominator $1 - \log P_{\mathrm{V}}(T \mid x)$ is at least one. This preserves probability ordering among compatible sequences and ranks them above incompatible ones. Exactness requires the returned minimum-free-energy pair set to equal T.

Structural agreement and exact coverage. We evaluate structural fold-back rather than recovery of a particular reference sequence (Joshi et al., 2025b). Pair Jaccard similarity diagnoses partial progress by penalizing missing and unwanted pairs:

$$J(x,T) = \frac{|F(x) \cap T|}{|F(x) \cup T|}. \tag{10}$$

We set $J(x,T) = 1$ when both pair sets are empty. Our primary endpoint is exact recovery, $F(x) = T$, equivalent to $J = 1$ under this convention. For a panel of N targets, define the first exact-solve

time $t_j^\star$ as elapsed wall time after initialization for target j, with $t_j^\star = 0$ for initial solves and $t_j^\star = \infty$ for completed nonsolves. At elapsed search time t and time horizon $t_{\max} > 0$, we report

$$\text{Coverage}(t) = \frac{1}{N} \sum_{j=1}^{N} \mathbf{1}[t_j^\star \le t], \tag{11}$$

$$\text{AUC}(t_{\max}) = \frac{1}{t_{\max}} \int_0^{t_{\max}} \text{Coverage}(t)\, dt. \tag{12}$$

Coverage includes every target in the evaluated panel and counts initial solves. The horizon is 30 minutes for the pseudoknot benchmarks and 60 minutes for the planar benchmarks and ablations. AUC rewards earlier recovery within the same time horizon. Incomplete panels yield coverage lower bounds, with unfinished or failed targets reported separately rather than treated as nonsolves. Final AUC and comparisons require complete, valid panels. Sequence composition and diversity are characterized in Appendix D.

H.2 Three-dimensional fold-back evaluation

Conditioning and sampling. KnitNet and Struct2SeQ-OA condition on the target's DSSR pair graph (Lu et al., 2015); the tertiary generators condition on 3D coordinates. We use their pretrained checkpoints and the settings in Table 8. For each of seeds 42, 731, and 2718, we generate a fresh 96-sequence pool for every generator, evaluate it as the frozen arm, and use that same pool to initialize the corresponding search arm. Generation and search randomness vary with the run seed. Generator weights remain fixed during search.

Table 8: Sampling settings for the 96 initial sequences per target.

Generator	Sampling
KnitNet	32 diffusion steps, temperature 0.8, top-$k = 2$, confidence-ordered pairwise commits with Watson–Crick and wobble pairs, final argmax.
Struct2SeQ-OA	Random decoding orders. 32 samples each from ϵ-greedy decoding at $\epsilon = 0.05$ and 0.10, and Q-softmax decoding.
gRNAde	Autoregressive sampling at temperature 0.1.
NA-MPNN	Random decoding orders at temperature 0.1.
RIDE	50 stochastic DDIM steps at temperature 0.1.
RiboDiffusion	50 ancestral diffusion steps with a cosine variance-preserving schedule and self-conditioning, using EMA weights.

Search objective and budgets. We retain RibonanzaNet-SS pairing residuals for repair construction and local surrogate learning, but use RhoFold+ to guide population ranking and terminal-pool selection toward tertiary agreement (Shen et al., 2024). Search and RL share the released RIDER reward $R : \mathcal{A}^L \to \mathbb{R}$ (Hu et al., 2026):

$$R(x) = 25g^2 - 0.25r^2 + \begin{cases} 100(g - 0.45), & g > 0.45, \\ 20(3 - r), & g \le 0.45 \text{ and } r < 3, \\ 0, & \text{otherwise.} \end{cases} \tag{13}$$

With fixed residue correspondences and a proper Kabsch fit on observed reference C4′ atoms, the distance-threshold agreement g averages the fractions at distances below 1, 2, 4, and 8 Å. The RMSD r is measured in Å. These released bonus thresholds do not define exact tertiary recovery. Optimization accesses the target coordinates; changing to C3′ for final scoring does not make evaluation independent.

Search allows 2,048 post-initialization logical evaluations plus up to 96 root evaluations. Its retained stopping rule can terminate when all active candidates match the secondary pair graph, without implying tertiary recovery. We export up to 96 distinct evaluated sequences ranked by $R(x)$, without forced retention of roots. RIDER instead restarts from RIDE for each target, uses 2,048 adaptation reward calls ($34 \times 60 + 8$ rollouts), then generates 96 candidates. Terminal slot counts therefore match, but total design-time allowances differ; final evaluation folds are additional for every arm.

RIDER adaptation uses 30 diffusion steps, temperature 0.5, learning rate 5×10^{-5}, policy clipping range 0.5, gradient-norm limit 1.0, two policy passes per batch, and mixed precision. Final sampling uses the RIDE settings in Table 8.

Final scoring and aggregation. We fold terminal candidates with a common RhoFold+ checkpoint in single-sequence mode, without MSA search, templates, or relaxation. US-align fits the RNA TM objective on the observed reference C3′ positions $\mathcal{O}_{C3'} \subseteq \{1, \ldots, L\}$ with fixed residue correspondences (Zhang et al., 2022):

$$\mathrm{scTM}(x) = \frac{1}{L} \sum_{i \in \mathcal{O}_{C3'}} \frac{1}{1 + (d_i / d_{0,\mathrm{RNA}}(L))^2}. \tag{14}$$

The aligned distance d_i between predicted and reference atoms at position i is measured in Å after US-align's heuristic rigid-body fit. The RNA length scale $d_{0,\mathrm{RNA}}(L)$ is also measured in Å. Both this scale and the normalization denominator use the original target length L, so missing atoms cannot shorten the normalization.

We select the highest-scTM candidate among each target's 96 slots, breaking ties by sample order. RMSD and $\mathrm{GDT_{TS}}$ describe that same candidate under a separate proper C3′ Kabsch fit; $\mathrm{GDT_{TS}}$ averages fractions within 1, 2, 4, and 8 Å. We retain duplicate native draws and failed or unfilled slots without replacement. Targets with no scored candidate receive zero scTM and undefined RMSD/$\mathrm{GDT_{TS}}$, omitted from those means. All 39 method–seed runs have scored candidates for all 64 targets, so every seed-level mean weights the full panel equally. We average each metric over the 64 targets within each seed, then report the mean and sample standard deviation of those averages over seeds 42, 731, and 2718.

Small-budget runtime comparison. We time sequential seed-42 runs on the same H200 GPU with eight CPU cores, using 3GES chain A (67 nt), the lower median target sorted by length and identifier. Each method receives 240 logical reward calls: 96 root plus 144 search calls, or four 60-rollout RIDER epochs with eight policy updates. Process-launch-to-exit timing includes loading, reward evaluation, updates, and output writing; it excludes root generation, RL terminal export, and final fold-back evaluation. Native caching remains enabled, so this measures total runtime rather than isolated optimizer overhead. It does not establish a full-budget or benchmark-wide speedup.

Shared computation accounting. Across both regimes, logical calls include inner-repair and cache-served requests. We count all evaluations executed in a batch and record the first exact candidate's ordered call index. Reported solve AUC uses elapsed wall time. We distinguish initialization from search/adaptation and final evaluation, and report physical oracle evaluations, neural forwards, and wall time because logical-call matching alone does not establish equal computational cost.

I LIMITATIONS AND FUTURE WORK

Wet-lab validation. Our study establishes computational generalization across several RNA inverse-design settings via exact fold-back matches and tertiary self-consistency measure agreement with computational oracles. While this is appropriate for the computational contribution we are submitting to this venue, we acknowledge that experimental structural validation and functional assays are essential to establish KnitNet's practical utility in the laboratory and assess its potential for integration into drug-development pipelines. We intend to participate in future OpenKnot challenges to evaluate our designs experimentally, following the chemical-mapping and structural-validation framework demonstrated by Townley et al. (2026). These experiments would test how improvements under our folding objectives translate to physical RNA structures.

Optimizing biological properties. Having demonstrated KnitNet's ability to optimize sequences under different structural objectives and target distributions, a natural next step is to replace or augment the folding-based objective with additional biological properties, such as binding, stability, or translation. Predictors trained on SELEX measurements or RNA–ligand binding labels could guide aptamer optimization, building on activity-guided generation such as RaptGen (Iwano et al., 2022). Other directions include RNA–protein binding, mRNA stability, and translation through coding-sequence and UTR design (Zhang et al., 2023; Sample et al., 2019). These applications would combine structural requirements with functional objectives and sequence constraints, such as preserving the encoded protein. Adapting repair to scalar biological feedback would also require learning which positions to modify when explicit pairing discrepancies are unavailable.

Preserving functional motifs. A central challenge in de novo functional RNA design is to combine sequence exploration with control over motifs essential to the intended biological activity (Joshi et al., 2025a; Townley et al., 2026). High native-sequence recovery is often used to assess RNA inverse folding: for example, gRNAde reports 56% mean recovery on its 14-target 3D benchmark (Joshi et al., 2025b). When examining our method's native-sequence recovery on OpenKnot57, we observed that the generator produces sequences with an average native-sequence recovery of 32.67%, which KnitNet search slightly reduces to 31.98%.These figures come from illustrate why native recovery should be distinguished from structural fidelity. Successful designs can differ substantially from a reference sequence, consistent with the sequence exploration reported in Struct2SeQ (He & Sun, 2026) and demonstrated experimentally in OpenKnot (Townley et al., 2026) This freedom supports de novo exploration, but functional motifs may need to remain intact. Both stages of KnitNet offer a route to preserving them: masked generation can condition on prescribed nucleotides, and joint repair already modifies selected positions while preserving the surrounding sequence. These capabilities could support design tasks where known functional motifs must be retained while the rest of the RNA is redesigned

Designing conformational behavior. Our current endpoint specifies one target structure. Although ViennaRNA guidance accounts for competing folds through target ensemble probability, we do not explicitly evaluate folding kinetics or robustness across conditions. Extending repair to desired and undesired conformational states could support RNA switches and other dynamic constructs, building on multi-state design in gRNAde and NUPACK (Joshi et al., 2025b; Wolfe et al., 2017). This would let search optimize a specified pattern of structural behavior rather than agreement with a single fold.

Richer structural feedback. Our 3D extension already benefits from tertiary-similarity ranking, while 2D pairing discrepancies guide repair. Geometric errors in residue distances, orientations, or contacts could provide more direct guidance for correcting tertiary packing. Richer interaction graphs could additionally represent noncanonical contacts and molecular interfaces beyond the current canonical/wobble pair assignments. Together, these extensions could connect joint repair to backbone and complex-design pipelines such as RFDpoly (Favor et al., 2025).

Scaling to longer RNAs. Repeated full-fold evaluations remain a computational cost, and our 512-nucleotide pretraining limit leaves kilobase-scale design unexplored. Hierarchical repair, inexpensive screening before full verification, and adaptive allocation between generation and search could extend practical reach. Differentiable folding methods such as JAX-RNAfold (Krueger & Ward, 2025) could also supply proposals where their folding assumptions apply, with complete-sequence verification retained to assess interactions between repaired regions.

J Full Pseudoknot Benchmark Results

Table 9: **Pseudoknot benchmark results.** Entries are mean ± sample SD over seeds 42, 731, and 2718. Exact is the endpoint number of solved targets; solve AUC is normalized over the 30-minute post-initialization horizon. Jaccard and MCC are endpoint structural diagnostics. Capped TTS is $\mathrm{mean}[\min(T_{\mathrm{first\ exact}}, 30\,\mathrm{min})]$ over the full panel, in minutes. Native initialization is untimed. The continued-sampling rows extend the same frozen native-initialization candidate pool using the source generator; this matched control was registered for Struct2Seq, Struct2Seq OA, and KnitNet. "Original implementation" means that the method's native structural oracle supplies online search guidance and RibonanzaNet-SS (RNSS) is used only for the common benchmark endpoint. "Adapted to RNSS" denotes our in-house variant in which RNSS also supplies online search guidance; these variants are not the authors' released implementations.

Method	Variant	Exact ↑	Solve AUC ↑	Jaccard ↑	MCC ↑	Capped TTS (min) ↓
A. OpenKnot57 (57 targets)						
A.1 Generator baselines						
Struct2Seq	Generation only	30.0 ± 0.00	–	0.9683 ± 0.0016	0.9579 ± 0.0049	–
	+ Continued sampling	44.7 ± 0.58	0.7221 ± 0.0199	0.9913 ± 0.0004	0.9884 ± 0.0003	8.34 ± 0.60
	+ KnitNet search	56.0 ± 0.00	0.9583 ± 0.0040	0.9989 ± 0.0006	0.9986 ± 0.0003	1.25 ± 0.12
Struct2Seq OA	Generation only	40.3 ± 1.53	–	0.9796 ± 0.0017	0.9724 ± 0.0006	–
	+ Continued sampling	49.7 ± 1.15	0.8346 ± 0.0105	0.9948 ± 0.0004	0.9923 ± 0.0003	4.96 ± 0.31
	+ KnitNet search	56.3 ± 0.58	0.9754 ± 0.0032	0.9996 ± 0.0004	0.9996 ± 0.0004	0.74 ± 0.10
gRNAde 2D	Generation only	8.7 ± 2.52	–	0.8744 ± 0.0101	0.8347 ± 0.0099	–
	+ KnitNet search	52.7 ± 1.15	0.8740 ± 0.0221	0.9899 ± 0.0025	0.9874 ± 0.0034	3.78 ± 0.66
RNAinformer	Generation only	9.7 ± 3.51	–	0.7313 ± 0.0087	0.7113 ± 0.0120	–
	+ KnitNet search	44.3 ± 0.58	0.7608 ± 0.0064	0.7886 ± 0.0009	0.7886 ± 0.0010	7.18 ± 0.19
A.2 Iterative solvers						
DesiRNA (original implementation)	Original init.	40.7 ± 1.15	0.6869 ± 0.0223	0.9807 ± 0.0013	0.9725 ± 0.0018	9.39 ± 0.67
	KnitNet init.	41.0 ± 2.65	0.6824 ± 0.0436	0.9826 ± 0.0027	0.9750 ± 0.0035	9.53 ± 1.31
DesiRNA (adapted to RNSS)	Original init.	54.3 ± 1.15	0.9192 ± 0.0170	0.9976 ± 0.0002	0.9971 ± 0.0004	2.42 ± 0.51
	KnitNet init.	55.0 ± 2.00	0.9306 ± 0.0148	0.9987 ± 0.0011	0.9984 ± 0.0014	2.08 ± 0.44
aRNAque (original implementation)	Original init.	23.0 ± 2.65	0.2748 ± 0.0448	0.7099 ± 0.0048	0.6957 ± 0.0020	21.75 ± 1.34
	KnitNet init.	36.0 ± 2.00	0.6005 ± 0.0278	0.8433 ± 0.0010	0.8376 ± 0.0016	11.98 ± 0.84
aRNAque (adapted to RNSS)	Original init.	49.7 ± 0.58	0.7392 ± 0.0144	0.9256 ± 0.0030	0.9234 ± 0.0029	7.83 ± 0.43
	KnitNet init.	53.0 ± 0.00	0.8916 ± 0.0081	0.9298 ± 0.0000	0.9298 ± 0.0000	3.25 ± 0.24
antaRNA (adapted to RNSS)†	Original init.	16.3 ± 3.21	0.2354 ± 0.0355	0.8924 ± 0.0028	0.8587 ± 0.0038	22.94 ± 1.06
MODENA (adapted to RNSS)†	Original init.	48.0 ± 2.00	0.8076 ± 0.0193	0.9933 ± 0.0028	0.9903 ± 0.0034	5.77 ± 0.58
Enzymer (original implementation)	Original init.	5.0 ± 1.73	0.0552 ± 0.0135	0.3473 ± 0.0276	0.3392 ± 0.0271	28.34 ± 0.40
	KnitNet init.	21.0 ± 1.00	0.3605 ± 0.0211	0.5176 ± 0.0010	0.5153 ± 0.0019	19.18 ± 0.63
PKProbDesign (original implementation)†	Original init.	35.7 ± 1.53	0.5896 ± 0.0169	0.7843 ± 0.0006	0.7822 ± 0.0004	12.31 ± 0.51
A.3 Our method						
KnitNet (ours)	Generation only	31.3 ± 1.53	–	0.9587 ± 0.0029	0.9455 ± 0.0026	–
	+ Continued sampling	45.0 ± 1.73	0.7326 ± 0.0149	0.9892 ± 0.0006	0.9851 ± 0.0006	8.02 ± 0.45
	+ KnitNet search	56.3 ± 1.15	0.9677 ± 0.0191	0.9996 ± 0.0008	0.9996 ± 0.0007	0.97 ± 0.57
B. PseudoBase++-339 (339 targets)						
B.1 Generator baselines						
Struct2Seq	Generation only	265.0 ± 3.61	–	0.9833 ± 0.0015	0.9783 ± 0.0022	–
	+ Continued sampling	320.3 ± 1.53	0.9211 ± 0.0059	0.9968 ± 0.0003	0.9966 ± 0.0003	2.37 ± 0.18
	+ KnitNet search	335.7 ± 0.58	0.9882 ± 0.0019	0.9993 ± 0.0001	0.9993 ± 0.0000	0.35 ± 0.06
Struct2Seq OA	Generation only	282.7 ± 4.16	–	0.9893 ± 0.0011	0.9871 ± 0.0015	–
	+ Continued sampling	331.0 ± 1.00	0.9622 ± 0.0014	0.9984 ± 0.0001	0.9983 ± 0.0001	1.13 ± 0.04
	+ KnitNet search	335.3 ± 0.58	0.9879 ± 0.0013	0.9993 ± 0.0001	0.9992 ± 0.0001	0.36 ± 0.04
gRNAde 2D	Generation only	175.0 ± 1.73	–	0.9273 ± 0.0013	0.9179 ± 0.0006	–
	+ KnitNet search	331.3 ± 1.53	0.9649 ± 0.0038	0.9970 ± 0.0000	0.9970 ± 0.0003	1.05 ± 0.11
RNAinformer	Generation only	178.7 ± 5.03	–	0.9391 ± 0.0029	0.9315 ± 0.0040	–
	+ KnitNet search	332.3 ± 2.89	0.9676 ± 0.0085	0.9951 ± 0.0003	0.9952 ± 0.0003	0.97 ± 0.25
B.2 Iterative solvers						
DesiRNA (original implementation)	Original init.	288.3 ± 4.04	0.8300 ± 0.0119	0.9878 ± 0.0015	0.9861 ± 0.0008	5.10 ± 0.36
	KnitNet init.	300.3 ± 2.89	0.8719 ± 0.0045	0.9917 ± 0.0003	0.9901 ± 0.0003	3.84 ± 0.13
DesiRNA (adapted to RNSS)	Original init.	334.7 ± 0.58	0.9805 ± 0.0031	0.9992 ± 0.0001	0.9991 ± 0.0002	0.59 ± 0.09
	KnitNet init.	335.0 ± 0.00	0.9853 ± 0.0003	0.9993 ± 0.0001	0.9991 ± 0.0002	0.44 ± 0.01
aRNAque (original implementation)	Original init.	185.3 ± 4.04	0.4743 ± 0.0059	0.9094 ± 0.0041	0.9031 ± 0.0051	15.77 ± 0.18
	KnitNet init.	301.7 ± 6.03	0.8685 ± 0.0150	0.9845 ± 0.0016	0.9839 ± 0.0016	3.94 ± 0.45

Continued on next page

Table 9 continued from previous page

Method	Variant	Exact ↑	Solve AUC ↑	Jaccard ↑	MCC ↑	Capped TTS (min) ↓
aRNAque (adapted to RNSS)	Original init.	334.0 ± 0.00	0.9546 ± 0.0020	0.9931 ± 0.0002	0.9930 ± 0.0003	1.36 ± 0.06
	KnitNet init.	333.3 ± 0.58	0.9801 ± 0.0013	0.9934 ± 0.0001	0.9933 ± 0.0001	0.60 ± 0.04
antaRNA (adapted to RNSS)†	Original init.	228.0 ± 3.61	0.6401 ± 0.0134	0.9494 ± 0.0012	0.9419 ± 0.0011	10.80 ± 0.40
MODENA (adapted to RNSS)†	Original init.	326.3 ± 0.58	0.9402 ± 0.0027	0.9980 ± 0.0003	0.9974 ± 0.0003	1.79 ± 0.08
Enzymer (original implementation)	Original init.	204.7 ± 3.51	0.5376 ± 0.0122	0.8054 ± 0.0088	0.7983 ± 0.0081	13.87 ± 0.37
	KnitNet init.	279.3 ± 0.58	0.8162 ± 0.0026	0.8853 ± 0.0003	0.8845 ± 0.0011	5.51 ± 0.08
PKProbDesign (original implementation)†	Original init.	294.0 ± 1.00	0.8462 ± 0.0019	0.8870 ± 0.0001	0.8869 ± 0.0001	4.61 ± 0.06
B.3 Our method						
KnitNet (ours)	Generation only	273.7 ± 5.03	–	0.9823 ± 0.0013	0.9796 ± 0.0016	–
	+ Continued sampling	322.3 ± 1.53	0.9384 ± 0.0032	0.9972 ± 0.0001	0.9965 ± 0.0003	1.85 ± 0.10
	+ KnitNet search	335.7 ± 0.58	0.9885 ± 0.0021	0.9993 ± 0.0002	0.9992 ± 0.0003	0.34 ± 0.06

† The released antaRNA, MODENA, and PKProbDesign implementations expose no legitimate external-sequence initialization interface. Their native results are retained as source-faithful baselines. All TTS values measure post-initialization search, not strict process-launch-to-answer wall time.

K Full Planar RNA Structure Benchmark Results

Table 10: **Complete ViennaRNA benchmark results and matched native-generation controls.** Entries are mean ± sample SD over seeds 42, 731, and 2718. Exact is the endpoint number of ViennaRNA-2.6.4 MFE-exact targets. Solve AUC is the normalized continuous first-exact solve curve over the 60-minute post-initialization horizon; equivalently, it is $1 - \mathrm{mean}[\min(T_{\text{first exact}}, 60\,\text{min})]/60\,\text{min}$. Jaccard and MCC are endpoint structural diagnostics where terminal sequences are available; dashes indicate unavailable measurements. Capped TTS is the corresponding full-panel mean time to first exact, in minutes. Native B96 is an untimed initialization result. The native-generation rows continue the identical frozen B96 pool using the source generator; this matched control was registered for Struct2SeQ, Struct2SeQ OA, RIFT-VAE, and KnitNet.

Method	Variant	Exact ↑	Solve AUC ↑	Jaccard ↑	MCC ↑	Capped TTS (min) ↓
A. Eterna100 (100 targets)						
A.1 Generator baselines						
Struct2SeQ	Generation only	33.0 ± 1.00	–	0.8093 ± 0.0058	0.8619 ± 0.0059	–
	+ Continued sampling	49.7 ± 0.58	0.4718 ± 0.0043	0.8846 ± 0.0030	0.9123 ± 0.0042	31.69 ± 0.26
	+ KnitNet search	81.0 ± 0.00	0.7620 ± 0.0047	0.9723 ± 0.0018	0.9780 ± 0.0020	14.28 ± 0.28
Struct2SeQ OA	Generation only	40.0 ± 1.73	–	0.8136 ± 0.0041	0.8535 ± 0.0044	–
	+ Continued sampling	56.7 ± 0.58	0.5421 ± 0.0095	0.9000 ± 0.0032	0.9185 ± 0.0035	27.47 ± 0.57
	+ KnitNet search	82.0 ± 1.00	0.7781 ± 0.0042	0.9670 ± 0.0022	0.9727 ± 0.0015	13.32 ± 0.25
RIFT-VAE	Generation only	38.0 ± 0.00	–	0.7521 ± 0.0000	0.8452 ± 0.0000	–
	+ Continued sampling	59.7 ± 0.58	0.5343 ± 0.0008	0.8816 ± 0.0006	0.9305 ± 0.0005	27.94 ± 0.05
	+ KnitNet search	85.7 ± 1.53	0.8061 ± 0.0076	0.9805 ± 0.0012	0.9811 ± 0.0008	11.63 ± 0.46
A.2 Iterative solvers						
DesiRNA	Original init.	87.7 ± 2.52	0.8461 ± 0.0184	0.9936 ± 0.0008	0.9944 ± 0.0011	9.23 ± 1.11
	KnitNet init.	80.7 ± 2.08	0.7773 ± 0.0160	0.9627 ± 0.0018	0.9720 ± 0.0021	13.36 ± 0.96
DRAG†	Original init.	62.0 ± 1.00	0.6015 ± 0.0119	0.9327 ± 0.0015	0.9421 ± 0.0022	23.91 ± 0.71
FastDesign	Original init.	80.3 ± 1.53	0.7793 ± 0.0147	0.9816 ± 0.0009	0.9856 ± 0.0006	13.24 ± 0.88
	KnitNet init.	75.3 ± 3.21	0.7373 ± 0.0333	0.9744 ± 0.0004	0.9803 ± 0.0005	15.76 ± 2.00
LEARNA†	Original init.	56.0 ± 1.73	0.5039 ± 0.0117	0.8581 ± 0.0064	0.8786 ± 0.0077	29.77 ± 0.70
Meta-LEARNA†	Original init.	62.3 ± 1.15	0.5980 ± 0.0029	0.9268 ± 0.0018	0.9549 ± 0.0015	24.12 ± 0.17
Meta-LEARNA-Adapt†	Original init.	63.3 ± 1.53	0.5962 ± 0.0051	0.9234 ± 0.0034	0.9544 ± 0.0025	24.23 ± 0.31
SamplingDesign†	Original init.	67.3 ± 1.15	0.6215 ± 0.0126	0.9034 ± 0.0031	0.9461 ± 0.0021	22.71 ± 0.76
GREED-RNA†	Original init.	80.7 ± 1.15	0.7821 ± 0.0096	0.9846 ± 0.0016	0.9853 ± 0.0017	13.07 ± 0.57
SAMFEO†	Original init.	80.3 ± 2.08	0.7678 ± 0.0102	0.9862 ± 0.0031	0.9875 ± 0.0017	13.93 ± 0.61
aRNAque	Original init.	72.0 ± 2.00	0.6970 ± 0.0120	0.9582 ± 0.0045	0.9721 ± 0.0024	18.18 ± 0.72
	KnitNet init.	74.0 ± 0.00	0.7184 ± 0.0038	0.9696 ± 0.0017	0.9777 ± 0.0029	16.90 ± 0.23
NEMO	Original init.	89.0 ± 1.00	0.8579 ± 0.0060	0.9893 ± 0.0010	0.9915 ± 0.0007	8.53 ± 0.36
	KnitNet init.	89.7 ± 0.58	0.8620 ± 0.0055	0.9897 ± 0.0011	0.9917 ± 0.0002	8.28 ± 0.33
SentRNA†	Original init.	51.7 ± 1.15	0.4548 ± 0.0122	0.8715 ± 0.0034	0.9167 ± 0.0057	32.71 ± 0.73
RNAiFold-LNS†	Original init.	66.7 ± 1.15	0.6378 ± 0.0072	–	–	21.73 ± 0.43
A.3 Our method						
KnitNet (ours)	Generation only	36.0 ± 1.73	–	0.8292 ± 0.0064	0.8810 ± 0.0093	–
	+ Continued sampling	59.7 ± 2.08	0.5684 ± 0.0074	0.9176 ± 0.0031	0.9426 ± 0.0004	25.90 ± 0.44
	+ KnitNet search	82.0 ± 1.00	0.7826 ± 0.0024	0.9789 ± 0.0025	0.9819 ± 0.0022	13.05 ± 0.14
B. Rfam-Learn-Test (100 targets)						
B.1 Generator baselines						
Struct2SeQ	Generation only	18.0 ± 3.61	–	0.9739 ± 0.0005	0.9668 ± 0.0003	–
	+ Continued sampling	62.7 ± 0.58	0.5154 ± 0.0248	0.9942 ± 0.0006	0.9925 ± 0.0008	29.07 ± 1.49
	+ KnitNet search	100.0 ± 0.00	0.9803 ± 0.0008	1.0000 ± 0.0000	1.0000 ± 0.0000	1.18 ± 0.05
Struct2SeQ OA	Generation only	65.0 ± 2.65	–	0.9926 ± 0.0003	0.9900 ± 0.0002	–
	+ Continued sampling	96.0 ± 0.00	0.9404 ± 0.0061	0.9996 ± 0.0000	0.9995 ± 0.0000	3.57 ± 0.37
	+ KnitNet search	100.0 ± 0.00	0.9966 ± 0.0006	1.0000 ± 0.0000	1.0000 ± 0.0000	0.21 ± 0.04
RIFT-VAE	Generation only	94.0 ± 0.00	–	0.9983 ± 0.0000	0.9981 ± 0.0000	–
	+ Continued sampling	99.0 ± 0.00	0.9865 ± 0.0001	0.9999 ± 0.0000	0.9999 ± 0.0000	0.81 ± 0.01
	+ KnitNet search	100.0 ± 0.00	0.9997 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	0.02 ± 0.00
B.2 Iterative solvers						
DesiRNA	Original init.	100.0 ± 0.00	0.9827 ± 0.0015	1.0000 ± 0.0000	1.0000 ± 0.0000	1.04 ± 0.09
	KnitNet init.	100.0 ± 0.00	0.9707 ± 0.0011	1.0000 ± 0.0000	1.0000 ± 0.0000	1.76 ± 0.07
DRAG†	Original init.	100.0 ± 0.00	0.9956 ± 0.0004	1.0000 ± 0.0000	1.0000 ± 0.0000	0.26 ± 0.02
FastDesign	Original init.	100.0 ± 0.00	0.9654 ± 0.0010	1.0000 ± 0.0000	1.0000 ± 0.0000	2.07 ± 0.06
	KnitNet init.	100.0 ± 0.00	0.9722 ± 0.0003	1.0000 ± 0.0000	1.0000 ± 0.0000	1.67 ± 0.02
LEARNA†	Original init.	76.0 ± 2.00	0.6130 ± 0.0326	0.9959 ± 0.0006	0.9946 ± 0.0007	23.22 ± 1.96

Continued on next page

Table 10 continued from previous page

Method	Variant	Exact ↑	Solve AUC ↑	Jaccard ↑	MCC ↑	Capped TTS (min) ↓
Meta-LEARNA†	Original init.	88.3 ± 1.15	0.8075 ± 0.0124	0.9985 ± 0.0001	0.9980 ± 0.0001	11.55 ± 0.74
Meta-LEARNA-Adapt†	Original init.	89.0 ± 1.00	0.8097 ± 0.0149	0.9986 ± 0.0001	0.9982 ± 0.0001	11.42 ± 0.90
SamplingDesign†	Original init.	67.0 ± 1.73	0.5401 ± 0.0113	0.9928 ± 0.0006	0.9910 ± 0.0011	27.59 ± 0.68
GREED-RNA†	Original init.	99.7 ± 0.58	0.9787 ± 0.0043	1.0000 ± 0.0000	1.0000 ± 0.0001	1.28 ± 0.26
SAMFEO†	Original init.	99.7 ± 0.58	0.9837 ± 0.0056	0.9999 ± 0.0002	0.9999 ± 0.0002	0.98 ± 0.34
aRNAque	Original init.	99.3 ± 1.15	0.8937 ± 0.0078	0.9999 ± 0.0001	0.9999 ± 0.0002	6.38 ± 0.47
	KnitNet init.	100.0 ± 0.00	0.9700 ± 0.0008	1.0000 ± 0.0000	1.0000 ± 0.0000	1.80 ± 0.05
NEMO	Original init.	100.0 ± 0.00	0.9908 ± 0.0004	1.0000 ± 0.0000	1.0000 ± 0.0000	0.55 ± 0.03
	KnitNet init.	100.0 ± 0.00	0.9989 ± 0.0006	1.0000 ± 0.0000	1.0000 ± 0.0000	0.07 ± 0.04
SentRNA†	Original init.	91.7 ± 1.53	0.8151 ± 0.0249	0.9984 ± 0.0005	0.9978 ± 0.0006	11.10 ± 1.49
RNAiFold-LNS†	Original init.	97.3 ± 0.58	0.9584 ± 0.0096	–	–	2.49 ± 0.58
B.3 Our method						
KnitNet (ours)	Generation only	29.7 ± 1.53	–	0.9689 ± 0.0013	0.9628 ± 0.0014	–
	+ Continued sampling	74.7 ± 1.53	0.6764 ± 0.0070	0.9942 ± 0.0005	0.9928 ± 0.0006	19.42 ± 0.42
	+ KnitNet search	100.0 ± 0.00	0.9759 ± 0.0032	1.0000 ± 0.0000	1.0000 ± 0.0000	1.45 ± 0.19
C. Rfam-Taneda (29 targets)						
C.1 Generator baselines						
Struct2SeQ	Generation only	8.7 ± 1.15	–	0.9052 ± 0.0031	0.9214 ± 0.0024	–
	+ Continued sampling	16.0 ± 0.00	0.5127 ± 0.0157	0.9599 ± 0.0038	0.9682 ± 0.0025	29.24 ± 0.94
	+ KnitNet search	23.0 ± 0.00	0.7827 ± 0.0011	0.9946 ± 0.0002	0.9954 ± 0.0002	13.04 ± 0.06
Struct2SeQ OA	Generation only	17.0 ± 1.00	–	0.9434 ± 0.0041	0.9563 ± 0.0048	–
	+ Continued sampling	20.0 ± 0.00	0.6872 ± 0.0010	0.9881 ± 0.0012	0.9903 ± 0.0011	18.77 ± 0.06
	+ KnitNet search	23.7 ± 0.58	0.7982 ± 0.0176	0.9950 ± 0.0007	0.9957 ± 0.0007	12.11 ± 1.06
RIFT-VAE	Generation only	17.0 ± 0.00	–	0.9245 ± 0.0000	0.9459 ± 0.0000	–
	+ Continued sampling	21.0 ± 0.00	0.7091 ± 0.0007	0.9798 ± 0.0013	0.9836 ± 0.0011	17.45 ± 0.04
	+ KnitNet search	24.3 ± 0.58	0.7982 ± 0.0091	0.9953 ± 0.0002	0.9959 ± 0.0002	12.11 ± 0.55
C.2 Iterative solvers						
DesiRNA	Original init.	24.0 ± 0.00	0.8243 ± 0.0007	0.9945 ± 0.0009	0.9954 ± 0.0007	10.54 ± 0.04
	KnitNet init.	23.3 ± 1.15	0.7768 ± 0.0265	0.9946 ± 0.0011	0.9952 ± 0.0010	13.39 ± 1.59
DRAG†	Original init.	23.7 ± 0.58	0.8125 ± 0.0238	0.9938 ± 0.0003	0.9944 ± 0.0002	11.25 ± 1.43
FastDesign	Original init.	24.0 ± 0.00	0.8101 ± 0.0016	0.9953 ± 0.0004	0.9959 ± 0.0004	11.39 ± 0.10
	KnitNet init.	24.0 ± 0.00	0.8132 ± 0.0006	0.9947 ± 0.0002	0.9955 ± 0.0005	11.21 ± 0.03
LEARNA†	Original init.	21.3 ± 0.58	0.6877 ± 0.0087	0.9883 ± 0.0020	0.9920 ± 0.0023	18.74 ± 0.52
Meta-LEARNA†	Original init.	23.7 ± 0.58	0.7927 ± 0.0194	0.9934 ± 0.0005	0.9945 ± 0.0008	12.44 ± 1.17
Meta-LEARNA-Adapt†	Original init.	23.0 ± 0.00	0.7754 ± 0.0100	0.9930 ± 0.0002	0.9941 ± 0.0006	13.48 ± 0.60
SamplingDesign†	Original init.	22.3 ± 0.58	0.6904 ± 0.0086	0.9861 ± 0.0019	0.9895 ± 0.0010	18.58 ± 0.52
GREED-RNA†	Original init.	24.0 ± 0.00	0.8246 ± 0.0011	0.9951 ± 0.0005	0.9956 ± 0.0006	10.52 ± 0.07
SAMFEO†	Original init.	24.0 ± 0.00	0.8268 ± 0.0003	0.9906 ± 0.0015	0.9947 ± 0.0003	10.39 ± 0.02
aRNAque	Original init.	23.3 ± 0.58	0.7847 ± 0.0034	0.9937 ± 0.0017	0.9952 ± 0.0005	12.92 ± 0.20
	KnitNet init.	23.3 ± 0.58	0.7705 ± 0.0188	0.9945 ± 0.0004	0.9951 ± 0.0003	13.77 ± 1.13
NEMO	Original init.	24.7 ± 0.58	0.8399 ± 0.0130	0.9957 ± 0.0004	0.9963 ± 0.0004	9.61 ± 0.78
	KnitNet init.	24.0 ± 0.00	0.8264 ± 0.0006	0.9954 ± 0.0002	0.9960 ± 0.0002	10.42 ± 0.03
SentRNA†	Original init.	21.7 ± 1.53	0.7083 ± 0.0445	0.9878 ± 0.0013	0.9899 ± 0.0009	17.50 ± 2.67
RNAiFold-LNS†	Original init.	24.0 ± 0.00	0.8083 ± 0.0152	–	–	11.50 ± 0.91
C.3 Our method						
KnitNet (ours)	Generation only	11.0 ± 1.00	–	0.8804 ± 0.0120	0.8992 ± 0.0146	–
	+ Continued sampling	14.3 ± 1.53	0.4685 ± 0.0343	0.9537 ± 0.0031	0.9592 ± 0.0014	31.89 ± 2.06
	+ KnitNet search	23.7 ± 0.58	0.7780 ± 0.0228	0.9949 ± 0.0006	0.9955 ± 0.0006	13.32 ± 1.37

† The released DRAG, LEARNA, Meta-LEARNA, Meta-LEARNA-Adapt, SamplingDesign, GREED-RNA, SAMFEO, SentRNA, and RNAiFold-LNS implementations expose no admitted external-sequence initialization interface. Their native results are retained as source-faithful baselines. RNAiFold-LNS emits a terminal sequence only upon finding an exact solution, so Jaccard and MCC are unavailable for unsolved runs and are shown as dashes. DRAG-EVO is unavailable from the official source, and RNA-FMQA was not evaluated because the official Amplify credential was unavailable. All TTS values measure post-initialization search rather than strict process-launch-to-answer wall time.

L INFERENCE-TIME SCALING

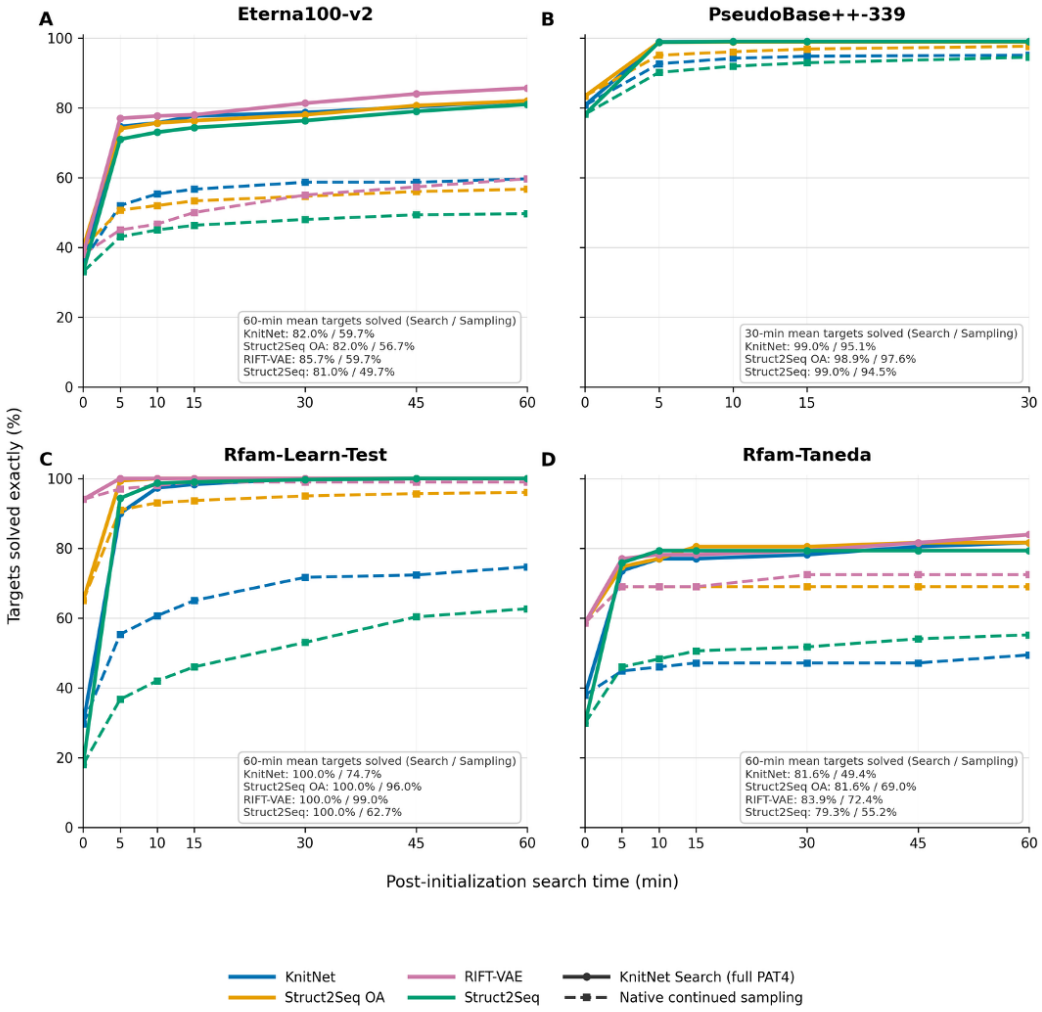

Figure 10: **KnitNet search converts additional inference time into additional exact solutions.** Exact coverage is measured against post-initialization wall-clock time from identical frozen B96 generator pools. Solid curves use KnitNet search, while dashed curves allocate the same time to native continued sampling. Curves report means over three seeds.

M MATH NOTATION

Table 11 summarizes the main notation for the KnitNet pipeline and evaluation.

Table 11: Principal notation for the KnitNet pipeline and evaluation.

Symbol	Meaning
Problem and initialization	
L, $\mathcal{A}$, $x \in \mathcal{A}^L$	Target length, RNA alphabet $\{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{U}\}$, and a candidate sequence
T	Target secondary structure as a set of nucleotide pairs, including crossing pairs
$F(x)$	Pair set predicted for x by the folding oracle
$E(x) = T \triangle F(x)$	Missing target pairs and unwanted predicted pairs
$\mathcal{R}$	Evaluated initial sequences sampled from the frozen generator
Counterexample-guided joint repair	
$\mathcal{A}_{\mathrm{pair}}$	Six pair-substitution values: AU, UA, GC, CG, GU, and UG
$\mathcal{P}$, $a \in \mathcal{P}$	Outer candidate population and the anchor of a repair episode
$\mathcal{V}_T$, $\mathcal{G}_a$, $\mathcal{E}_a$	Target-defined assignment units, their error graph, and its edges from anchor and ancestor errors
$S \subseteq \mathcal{V}_T$	Editable scope of assignment units. $\lvert S\rvert$ counts units
z, x	Joint nucleotide assignment on S and the resulting sequence, which matches a outside those units
C	Remembered, prioritized error-pair set in the current episode
$m_F(x)$	Oracle-specific merit used to rank evaluated sequences, with larger values preferred
$\ell_C(x)$	Lexicographic repair loss $\big(\lvert E(x) \cap C\rvert, \lvert E(x)\rvert, -m_F(x)\big)$
$\mathcal{D}$, $\mathcal{H} \subseteq \mathcal{D}$	Evaluated joint proposals in an episode, including its anchor, and the protected proposal beam
$\phi_S(x)$, $f(\phi_S(x))$	Scope-assignment features and the surrogate's predicted prioritized and total error counts
$\mathcal{Q}$, $\mathcal{J}, \mathcal{U}$	Joint proposals not yet evaluated in the episode, selected joint batch, and single-unit mutation batch. Batch sizes count evaluation requests
$x^\star$	Best evaluated proposal in a nonempty joint batch, whose errors update C
b, B	Logical folding evaluations spent after initialization and the allowed budget
Evaluation and 3D design	
$J(x, T)$	Pair-set Jaccard similarity of the predicted fold and target
N, $t_j^\star$, $\mathrm{Coverage}(t)$, $\mathrm{AUC}(t_{\max})$	Panel size, first exact-solve time, solve coverage, and its normalized area over time horizon $t_{\max}$
$R(x)$	Tertiary reward used to rank candidates during 3D search
$\mathrm{scTM}(x)$	RNA TM-score of the predicted 3D fold against the reference