Learning to Design RNA

Frederic Runge*, **Danny Stoll***, **Stefan Falkner & Frank Hutter**
Department of Computer Science, University of Freiburg
{runget,stolld,sfalkner,fh}@cs.uni-freiburg.de

Abstract

Designing RNA molecules has garnered recent interest in medicine and biotechnology since many functional RNA molecules were shown to be involved in regulatory processes for transcription, epigenetics and translation. Here, we propose a new algorithm for the *RNA Design* problem, dubbed *LEARNA*, which uses deep reinforcement learning to train a policy network to sequentially design an entire RNA molecule. By meta-learning across 8000 different RNA Design tasks, our extension *Meta-LEARNA* constructs an RNA Design policy that can be applied out of the box to quickly solve novel RNA Design tasks. Finally, we use an AutoML approach to jointly optimize over a rich space of neural architectures for the policy network, the hyperparameters of the training procedure and the formulation of the decision process. In a comprehensive empirical evaluation we transfer the found architectures and parameter settings to three qualitatively different benchmark sets of RNA Design tasks and show that our approach achieves new state-of-the-art performance on all benchmarks while also being orders of magnitudes faster in reaching the previous state-of-the-art performance.

1 Introduction

At its most basic form, RNA is a sequence of the four nucleotides *Adenine (A)*, *Guanine (G)*, *Cytosine (C)* and *Uracile (U)*. While the RNA sequence serves as the blueprint, the functional structure of the RNA molecule is determined by the folding translating the RNA sequence into its 3D tertiary structure. The hydrogen bonds formed between two corresponding nucleotides constitute one of the driving forces during this process; the structure that encompasses these hydrogen bonds is commonly referred to as the secondary structure of RNA.

The problem of finding an RNA sequence that folds into a desired secondary structure is known as the *RNA Design* problem or *RNA inverse folding* [Hofacker et al., 1994]. Most algorithms for *RNA Design* focus on search strategies that start with an initial nucleotide sequence and modify it to find a solution for the given secondary structure [Hofacker et al., 1994, Andronescu et al., 2004, Taneda, 2011, Esmaili-Taheri et al., 2014, Eastman et al., 2018]. In contrast, in this paper we describe a novel deep reinforcement learning (RL) approach to this problem. Our contributions are as follows:

- We describe *LEARNA*, a deep RL algorithm for RNA design. *LEARNA* trains a policy network that, given a target secondary structure, can be rolled out to sequentially predict the entire RNA sequence. After generating an RNA sequence, our approach folds this sequence, locally adapts it, and uses the distance of the resulting structure to the target structure as an error signal for the RL agent.
- We describe *Meta-LEARNA*, a meta-learning version of *LEARNA* that learns a single policy across many *RNA Design* problems directly applicable to new *RNA Design* problems. Specifically, it learns a conditional generative model from which we can sample candidate RNA sequences for a given target structure that solves many problems with the first sample.

*Frederic Runge and Danny Stoll contributed equally to this work; order determined by coinflip.

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

- We introduce a new benchmark dataset, which we used to develop and tune our algorithm.
- We apply an AutoML approach [Hutter et al., 2018] to jointly optimize the architecture of the policy network together with the training hyperparameters, and the state representation. To the best of our knowledge, this is the first application of a variant of neural architecture search to improve the performance of RL and meta-learning.
- A comprehensive empirical analysis shows that our approach achieves new state-of-the-art performance on the two most commonly used RNA design benchmark datasets: Rfam-Taneda (following Taneda [2011]) and Eterna100 (following Anderson-Lee et al. [2016]), as well as on the test split of our new benchmark. Furthermore, *Meta-LEARNA* achieves the results of the previous state-of-the-art approaches $26\times$, $450\times$ and $4\times$ faster.

2 Background and Related Work

The *RNA Design* problem aims to find an inverse mapping for a given RNA folding algorithm $\mathcal{F}$, which maps from an RNA sequence ϕ to a representation of its secondary structure: Given a folding algorithm $\mathcal{F}$ and a target RNA secondary structure ω, the *RNA Design* problem is to find an RNA sequence ϕ that satisfies $\omega = \mathcal{F}(\phi)$. This is illustrated in Figure 1 in Appendix B.

Most algorithms targeting the *RNA Design* problem are either (1) local approaches that operate on a single sequence and try to find a solution by changing a small number of nucleotides at a time with the loss function guiding the search (*RNA-SSD* [Andronescu et al., 2004], *INFO-RNA* [Busch and Backofen, 2006], *NUPACK* [Dirks and Pierce, 2004, Zadeh et al., 2010], *ERD* [Esmaili-Taheri et al., 2014] and the approach by Eastman et al. [2018]) or (2) global methods, either having a large number of candidates being manipulated, or modeling a global distribution from which samples are sampled (*MODENA* [Taneda, 2011], *AntaRNA* [Kleinkauf et al., 2015] and *MCTS-RNA* [Yang et al., 2017]). A more detailed review can be found in Churkin et al. [2017].

Other than ours, the only existing algorithm that uses RL is the one by Eastman et al. [2018]; this has been developed in parallel to our work and, in contrast to our end-to-end approach only uses RL to guide a local search, making it far less sample-efficient. In the remainder of the paper, we refer to this approach as RL-LS and compare to it, as well as the recent state-of-the-art algorithms *MCTS-RNA*, *AntaRNA*, and *RNAinverse*.

The work by Bello et al. [2016] heavily influenced our work. In it, the authors apply RL to combinatorial problems, namely the Traveling Salesman Problem. Inspired by this work, we propose to frame the *RNA Design* problem as a RL problem where the agent predicts which nucleotide to place next into the sequence, learning to design RNA end-to-end.

3 A Decision Process Modelling RNA Design

We propose to model the *RNA Design* problem with respect to a given target structure ω as the undiscounted Decision Process $D_\omega := (\mathcal{S}, \mathcal{A}, \mathcal{R}_\omega, \mathcal{P}_\omega)$. In our experiments, we used the popular *Zuker algorithm* [Zuker and Stiegler, 1981, Zuker and Sankoff, 1984] for folding, but our approach would directly be applicable to any other folding algorithm.

Action space In each episode an agent has the task to design an RNA sequence that folds into the given target structure ω. To design a candidate solution ϕ the agent places nucleotides by choosing an action a^t at each time step t. For unpaired sites, action a^t corresponds to selecting one of the four RNA nucleotides (G, C, A or U), while for paired sites, a^t corresponds to selecting one of the Watson-Crick base pairs (GC, CG, AU, or UA). This is illustrated in Figure 2 in Appendix B.

State space and Transition Function The agent chooses an action a^t based on the state s^t provided by the environment. We set s^t to the $(2\kappa + 1)$-gram centered around the t-th site of the target structure ω, where κ is a hyperparameter we dub the state radius. To be able to construct this centered n-gram at all sites we introduced κ padding characters at the start and the end of the target structure. Since at each time step t the state s^t is set to a fixed $(2\kappa + 1)$-gram, the transition function $\mathcal{P}_\omega$ is deterministic and defined accordingly.

Reward At the terminal time step T the agent has assigned nucleotides to all sites and the environment generates the (only non-zero) reward r^T:

$$r^T = -\left(\frac{\mathrm{d_H}(\mathrm{fold}(\phi),\, \omega)}{|\omega|}\right)^{\alpha} \quad , \tag{1}$$

where $\mathrm{d_H}(\cdot,\cdot)$ is the *Hamming distance*, $\mathrm{fold}(\cdot)$ denotes the Zuker folding algorithm and $\alpha > 1$ is a hyperparameter to shape the reward. If the distance between the fold of the proposed sequence and the target structure becomes small enough, i.e., $\mathrm{d_H}(\mathrm{fold}(\phi),\, \omega) < \xi$ (we set $\xi = 5$ based on preliminary experiments), we employ a local improvement step before computing the reward: this tries all nucleotide combinations for the sites at which the fold of our designed sequence $\mathrm{fold}(\phi)$ differs from the target structure ω.

4 Obtaining Policies for RNA Design

We propose several strategies to learn the parameters θ of a given policy network π^θ:

LEARNA The *LEARNA* strategy learns to design a sequence for the target structure ω in an online fashion, from scratch. The parameters θ of the policy network π^θ are randomly initialized before the agent episodically interacts with the decision process $\mathcal{D}_\omega$. For updating the parameters we use the policy gradient method PPO [Schulman et al., 2017].

Meta-LEARNA *Meta-LEARNA* uses a meta-learning approach [Lemke et al., 2015] that views the RNA design of the target structures in the training set Ω_{train} as tasks and learns to transfer knowledge across them. Each of the target structures $\omega_i \in \Omega_{train}$ defines a different decision process $\mathcal{D}_{\omega_i}$, and we train a single policy network on all of them, also using PPO. Once the training is finished, the parameters θ are fixed and π^θ can be applied to the decision process $\mathcal{D}_\omega$ defined by a new target structure ω by sampling from the learned generative model.

Meta-LEARNA-Adapt *Meta-LEARNA-Adapt* combines the previous two strategies: First, we run *Meta-LEARNA* to train parameters θ in an offline training phase on Ω_{train}. However, to work on target structure ω, the policy is not fixed but is only used to initialize *LEARNA* running on the Decision Process $\mathcal{D}_\omega$.

The precise architectures of the policy networks we used were determined using neural architecture search as described in the next section; they are listed in Table 5 in Appendix E.

5 Joint Architecture Search and Hyperparameter Optimization

To automatically select the best neural architecture based on data, we define a search space that includes both elements of convolutional neural networks (CNNs) and recurrent neural networks (RNNs). We construct our architectures as follows: (1) the dot bracket representation of the state is either binary encoded (distinguishing between paired and unpaired sites) or processed by an embedding layer that converts the symbol-based representation into a learnable numerical one for each site. Then, (2) an optional CNN with at most two layers can be applied to the state, followed by (3) an optional LSTM with at most two layers. As the final stage, we always add (4) a shallow fully-connected network with one or two layers, which outputs the distribution over actions. Jointly with these architectural choices, we optimized three hyperparameters of PPO (learning rate, batch size, and strength of the entropy regularization), and since we want to use the best decision process for solving our problem, also two hyperparameters of the decision process described in Section 3: the exponent α used for reward shaping and the state space radius κ. In total, these choices yield a 14-dimensional space comprising mostly integer variables; for full details see Appendix E.

We used the efficient Bayesian optimization method BOHB [Falkner et al., 2018] to address the problems of neural architecture search (NAS) [Zoph and Le, 2017, Elsken et al., 2018] and hyperparameter optimization as a joint optimization problem. BOHB supports optimization in mixed integer/continuous search spaces, can utilize parallel resources and additionally allows to exploit cheap approximations of expensive-to-evaluate objective functions to speed up the optimization. To avoid overfitting on our target RNA benchmarks, we further introduce a new training set (Rfam-LEARN-Train), validation set (Rfam-LEARN-Validation) and test set (Rfam-Learn-Test) described in detail

Table 1: Summary of results for *RNAInverse*, *MCTS-RNA*, *RL-LS*, *antaRNA*, *LEARNA*, and *Meta-LEARNA* regarding the total fraction of solved target structures on the three benchmarks. A target structure counts as solved if a solution was found in any of the evaluation runs on the specific dataset.

METHOD	ETERNA100	RFAM-TANEDA	RFAM-LEARN-TEST
MCTS-RNA	57%	79%	97%
ANTARNA	58%	66%	**100%**
RL-LS	59%	62%	62%
RNAINVERSE	60%	59%	95%
LEARNA-10MIN	-	79%	95%
LEARNA-30MIN	63%	-	97%
META-LEARNA-ADAPT	64%	**83%**	98%
META-LEARNA	**65%**	79%	**100%**

in Appendix C. We created two versions of *LEARNA*: (1) *LEARNA-10min*, which is optimized for achieving strong performance in 10 minutes (on one core per sequence) and (2) *LEARNA-30min*, which is optimized for achieving strong performance within 30 minutes (on one core per sequence). *LEARNA-10min* is used on the Rfam-Taneda dataset and *LEARNA-30min* is applied to the other two datasets. Finally, our meta learning approach *Meta-LEARNA* is optimized to achieve strong performance when run for one hour on twenty cores (with an internal budget of 1 minute on one core per sequence). Based on preliminary experiments, we used the sum of mean distances as the loss.

6 Experiments

We report results on two established benchmarks from the literature, the Rfam benchmark as described at Taneda [2011] and the Eterna100 benchmark [Anderson-Lee et al., 2016], as well as on our proposed benchmark Rfam-Learn, reporting the accumulated number of solved targets across all runs. Details on the benchmarks we used are listed in Appendix D. We used the same hardware for all approaches and experiments as listed in detail in Appendix A. The final performance of all algorithms of our comparison for the three benchmarks is summarized in Table 1. The corresponding plots for all benchmarks are shown in Appendix F. On the Eterna100 benchmark all variants of *LEARNA* achieve new state-of-the-art results. Remarkably, *Meta-LEARNA* achieves previous state-of-the-art performance in about 90 seconds, new state-of-the-art performance in less than 3 minutes and also achieves new state-of-the-art performance in each single run (solving at least 64 % of the target structures). Concerning the Rfam-Taneda benchmark, *Meta-LEARNA* and *LEARNA* are on par with the current state-of-the-art results of *MCTS-RNA*. Remarkably, *Meta-LEARNA* needs less than 10 seconds to achieve this performance. *Meta-LEARNA-Adapt* achieves new state-of-the-art results after 1 minute, solving 83% of the target structures. On our proposed Rfam-Learn benchmark, only *Meta-LEARNA* and *antaRNA* were able to solve all of the target structures in 1 hour; 5 minutes and 20 minutes respectively. Except for *RL-LS*, all algorithms could solve at least 95% of the target structures within the 1h time limit. The results of our ablation study are illustrated in the Appendix G. We observed that the local improvement step boosts performance for all variants of our approach. The performance loss of *Meta-LEARNA-Adapt* compared to *Meta-LEARNA* is caused by the overhead associated with the weight updates, which result in *Meta-LEARNA-Adapt* only being able to perform 7-10 times fewer evaluations than *Meta-LEARNA* performs in the same time.

7 Conclusion

We proposed the deep reinforcement learning algorithm *LEARNA* for the *RNA Design* problem to sequentially construct candidate solutions in an end-to-end fashion. By pre-training on a large corpus of biological sequences, a local improvement step to aid the agent, and extensive architecture and hyperparameter optimization, we arrived at *Meta-LEARNA*, a ready-to-use agent that achieves state-of-the-art results on the Eterna100 benchmark. By continuing training, dubbed *Meta-LEARNA-Adapt*, we can also improve over all previous results on the Rfam benchmark[2].

[2]Code and data for reproducing our results is available at `https://github.com/automl/learna`

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

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.

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. ISSN 0022-2836. doi: https://doi.org/10.1016/j.jmb.2003.12.041. URL `http://www.sciencedirect.com/science/article/pii/S0022283603015596`.

Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

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. URL `http://dx.doi.org/10.1093/bioinformatics/btl194`.

Alexander Churkin, Matan Drory Retwitzer, Vladimir Reinharz, Yann Ponty, Jérôme Waldispühl, and Danny Barash. Design of rnas: comparing programs for inverse rna folding. *Briefings in bioinformatics*, 19(2):350–358, 2017.

Robert M. Dirks and Niles A. Pierce. An Algorithm for Computing Nucleic Acid Base-Pairing Probabilities Including Pseudoknots. *Journal of Computational Chemistry*, 25(10):295—-1304, 2004. doi: 10.1002/jcc.20057.

Peter Eastman, Jade Shi, Bharath Ramsundar, and Vijay S Pande. Solving the RNA design problem with reinforcement learning. *PLoS computational biology*, 14(6):e1006176, 2018.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural Architecture Search: A Survey. *ArXiv e-prints*, August 2018.

Ali Esmaili-Taheri, Mohammad Ganjtabesh, and Morteza Mohammad-Noori. Evolutionary solution for the RNA design problem. *Bioinformatics*, 30(9):1250–1258, 2014. doi: 10.1093/bioinformatics/btu001. URL `http://dx.doi.org/10.1093/bioinformatics/btu001`.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, pages 1436–1445, July 2018.

Ivo Hofacker, Walter Fontana, Peter Stadler, Sebastian Bonhoeffer, Manfred Tacker, and Peter Schuster. Fast Folding and Comparison of RNA Secondary Structures. *Monatshefte fuer Chemie/Chemical Monthly*, 125:167–188, 02 1994.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 2018. In press, available at http://automl.org/book.

Ioanna Kalvari, Joanna Argasinska, Natalia Quinones-Olvera, Eric P Nawrocki, Elena Rivas, Sean R Eddy, Alex Bateman, Robert D Finn, and Anton I Petrov. Rfam 13.0: shifting to a genome-centric resource for non-coding RNA families. *Nucleic acids research*, 46(D1):D335–D342, 2017.

Robert Kleinkauf, Torsten Houwaart, Rolf Backofen, and Martin Mann. antaRNA–Multi-objective inverse folding of pseudoknot RNA using ant-colony optimization. *BMC bioinformatics*, 16(1): 389, 2015.

Christiane Lemke, Marcin Budka, and Bogdan Gabrys. Metalearning: a survey of trends and technologies. 44(1):117–130, Jun 2015.

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, Nov 2011a. ISSN 1748-7188. doi: 10.1186/1748-7188-6-26. URL https://doi.org/10.1186/1748-7188-6-26.

Ronny Lorenz, Stephan H Bernhart, Christian Hoener Zu Siederdissen, Hakim Tafer, Christoph Flamm, Peter F Stadler, and Ivo L Hofacker. Viennarna package 2.0. *Algorithms for Molecular Biology*, 6(1):26, 2011b.

Michael Schaarschmidt, Alexander Kuhnle, and Kai Fricke. Tensorforce: A tensorflow library for applied reinforcement learning. Web page, 2017. URL https://github.com/reinforceio/tensorforce.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Akito Taneda. MODENA: a multi-objective RNA inverse folding. *Advances and applications in bioinformatics and chemistry: AABC*, 4:1, 2011.

Xiufeng Yang, Kazuki Yoshizoe, Akito Taneda, and Koji Tsuda. RNA inverse folding using Monte Carlo tree search. *BMC bioinformatics*, 18(1):468, 2017.

Joseph N. Zadeh, Conrad D. Steenberg, Justin S. Bois, Brian R. Wolfe, Marshall B. Pierce, Asif R. Khan, Robert M. Dirks, and Niles A. Pierce. Nupack: Analysis and design of nucleic acid systems. *Journal of Computational Chemistry*, 32(1):170–173, 2010. doi: 10.1002/jcc.21596. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.21596.

B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017. Published online: iclr.cc.

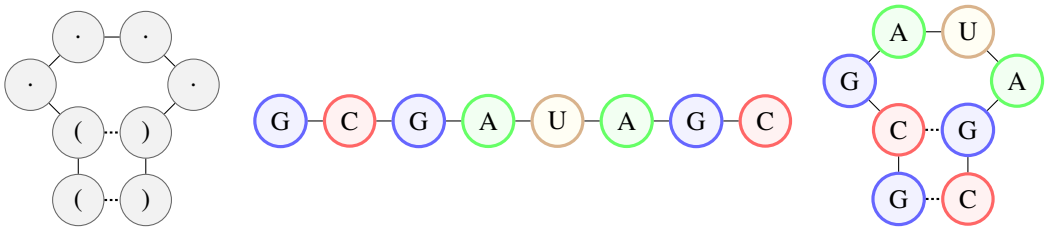
M Zuker and P. Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic Acids Research*, 9(1):133–148, 1981.

Michael Zuker and David Sankoff. RNA secondary structures and their prediction. *Bulletin of Mathematical Biology*, 46(4):591 – 621, 1984. ISSN 0092-8240. doi: https://doi.org/10.1016/S0092-8240(84)80062-2. URL http://www.sciencedirect.com/science/article/pii/S0092824084800622.

A Technical Details

We used the implementation of the *Zuker algorithm* provided by *ViennaRNA* [Lorenz et al., 2011b] versions 2.4.8 (*MCTS-RNA*, *RL-LS* and *LEARNA*), 2.1.9 (*AntaRNA*) and 2.4.9 (*RNAInverse*). Our implementation uses the Reinforcement Learning library *tensorforce*, version 0.3.3 [Schaarschmidt et al., 2017] working with *TensorFlow* version 1.4.0 [Abadi et al., 2015]. All computations were done on Broadwell E5-2630v4 2.2 GHz CPUs with 5 GByte RAM. For the training phase of *Meta-LEARNA*, we used all 20 cores of these machines, but at evaluation time, all methods were only allowed a single core (using core binding).

B Illustration



(a) Target structure ω (b) RNA sequence $\phi \in N^*$ (c) Fold $\mathcal{F}(\phi)$

Figure 1: Illustration of the *RNA Design* problem using a folding algorithm $\mathcal{F}$. RNA secondary structures are often represented using the dot-bracket notation, where dots stand for unbound sites and nucleotides connected by a hydrogen bond are marked by opening and closing brackets. Given the desired RNA secondary structure and its dot-bracket notation (a), the task is to design an RNA sequence (b) that folds into the desired secondary structure (c).

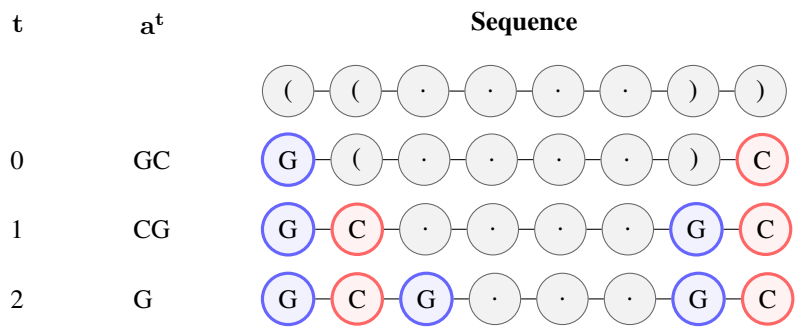


Figure 2: Illustration of the design of a candidate solution. Starting at time point 0 the candidate solution is sequentially built by placing nucleotides from the action sequence $(a^t)_{t \in \{0,\ldots,T-1\}}$.

C Rfam-Learn Dataset

To ensure a large enough and interesting dataset, we downloaded all families of the Rfam database version 13.0 [Kalvari et al., 2017] and folded them using the ViennaRNA package [Lorenz et al., 2011a]. We removed all secondary structure with multiple known solutions, and only kept structures with lengths between 50 and 450 to match the existing datasets. To focus on the harder sequences, we only kept the ones that a single run of *MCTS-RNA* could not solve within 30 seconds. We chose *MCTS-RNA* for filtering as it was the fastest algorithm from the literature. The remaining secondary structures were split into a training set of 65000, a validation set of 100, and a test set of 100 secondary structures.

D Benchmarks

Table 2: Overview on the three benchmarks Eterna100 [Anderson-Lee et al., 2016], Rfam-Taneda [Taneda, 2011] and Rfam-Learn we used for our experiments. The table displays the timeout, the number of evaluations for each target structure, the number of sequences and the range of sequence lengths for the corresponding benchmark.

DATASET	TIMEOUT	EVALUATIONS	SEQUENCES	LENGTH
ETERNA100	24H	5	100	12–400
RFAM-TANEDA	10MIN	50	29	54–451
RFAM-LEARN-TRAIN	–	–	65000	50–450
RFAM-LEARN-VAL	–	–	100	50–444
RFAM-LEARN-TEST	1H	5	100	50–446

E Architecture and Hyperparameter Search Space

Table 3: Search space for the agent's architecture and the trainings hyperparameter used for training *Meta-Learna*

Parameter Name	Type	Range	Prior
filter size in 1^{st} conv layer	integer	$[0, 8]$	uniform
filter size in 2^{nd} conv layer	integer	$[0, 4]$	uniform
# filters in 1^{st} conv layer	integer	$[1, 32]$	uniform
# filters in 2^{nd} conv layer	integer	$[1, 32]$	uniform
# fully connected layers	integer	$[1, 2]$	uniform
# units in fully connected layer(s)	integer	$[8, 64]$	log-uniform
# LSTM layers	integer	$[0, 2]$	uniform
# units in every LSTM layer	integer	$[1, 64]$	log-uniform
# state space radius	integer	$[0, 32]$	uniform
embedding dimensionality	integer	$[0, 4]$	uniform
batch size	integer	$[32, 128]$	log-uniform
entropy regularization	float	$[5 \cdot 10^{-5}, 5 \cdot 10^{-3}]$	log-uniform
learning rate for PPO	float	$[1 \cdot 10^{-6}, 1 \cdot 10^{-4}]$	log-uniform
reward exponent	float	$[1, 10]$	uniform

Table 4: Search space for the agent's architecture and the trainings hyperparameter used for training *LEARNA* for both the 10 and 30 minutes budget

Parameter Name	Type	Range	Prior
filter size in 1^{st} conv layer	integer	$[0, 8]$	uniform
filter size in 2^{nd} conv layer	integer	$[0, 4]$	uniform
# filter in 1^{st} conv layer	integer	$[1, 32]$	log-uniform
# filter in 2^{nd} conv layer	integer	$[1, 32]$	log-uniform
# fully connected layers	integer	$[1, 2]$	uniform
# units in fully connected layer(s)	integer	$[8, 64]$	log-uniform
# LSTM layers	integer	$[0, 2]$	uniform
# units in every LSTM layer	integer	$[1, 64]$	log-uniform
# state space radius	integer	$[1, 32]$	uniform
embedding dimensionality	integer	$[0, 4]$	uniform
batch size	integer	$[32, 128]$	log-uniform
entropy regularization	float	$[1 \cdot 10^{-5}, 1 \cdot 10^{-2}]$	log-uniform
learning rate for PPO	float	$[1 \cdot 10^{-5}, 1 \cdot 10^{-3}]$	log-uniform
reward exponent	float	$[1, 10]$	uniform

Table 5: The selected configurations for each scenario and budget.

Parameter Name	LEARNA-10min	LEARNA-30min	Meta-LEARNA
filter size in 1st conv layer	5	0	5
filter size in 2nd conv layer	3	3	7
# filters in 1st conv layer	8	10	32
# filters in 2nd conv layer	1	3	14
# fully connected layers	1	1	1
# units in fully connected layer(s)	52	32	9
# LSTM layers	2	2	0
# units in every LSTM layer	4	7	53
# state space radius	16	2	26
embedding dimensionality	0	0	1
batch size	32	79	80
entropy regularization	$4.44 \cdot 10^{-4}$	$1.63 \cdot 10^{-4}$	$1.98 \cdot 10^{-4}$
learning rate for PPO	$5.49 \cdot 10^{-4}$	$3.38 \cdot 10^{-4}$	$6.37 \cdot 10^{-5}$
reward exponent	5.72	9.43	9.22

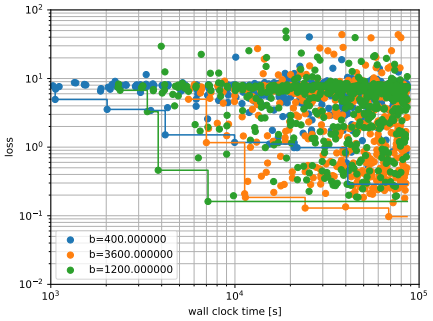


Figure 3: Observed validation loss during the BOHB run for *Meta-LEARNA*. The different budgets b correspond to the training time on 20 CPUs in parallel. The results seem to suggest that one can achieve a very similar performance with only 20 minutes of training, which could imply that much longer training of the agent might be required for substantially better performance.

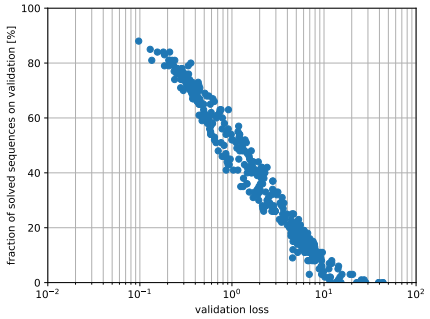


Figure 4: Relationship between the observed loss and the number of solved sequences. The plot suggests that our loss metric correlates strongly with the number of successfully found primary sequences.

F Comparison Plots

Here, we show the performance of all methods tested on all three benchmarks. In particular, we present the fraction of solved sequences for Eterna100, Rfam-Taneda, and Rfam-Learn-Test accumulated and averaged over independent evaluation runs.

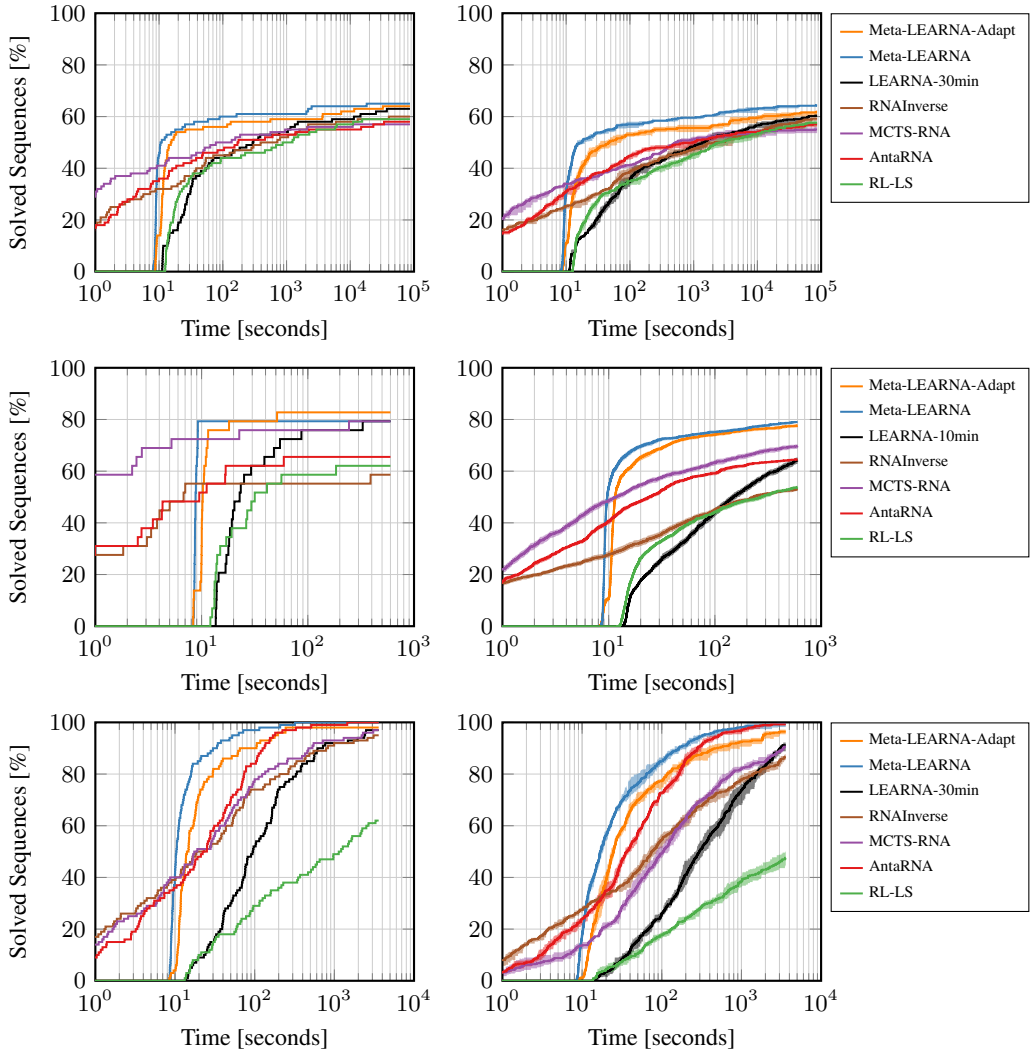


Figure 5: Comparison of all methods on the Eterna100 (top) Rfam-Taneda (middle) and Rfam-Learn-Test (bottom) benchmark. The left side shows the fraction of solved target structures accumulated over independent evaluation runs, while the right side shows the mean of that fraction with confident intervals using 5, 50, and 5 independent evaluation runs respectively. On all three benchmarks, Meta-LEARNA outperforms all other methods in terms of number of solved sequences and/or time to achieve this performance. For Eterna100, all three strategies of our novel approach achieve new state-of-the-art results, while being orders of magnitudes faster. On our benchmark, all algorithms except *RL-LS* solve at least 95 % of the target structures, but Meta-LEARNA performs best (after a short lag due to computational overhead).

G Ablation

Here, we study the contribution of different components of our approaches with an ablation. By removing one component at a time, we can see the impact it has on the final performance.

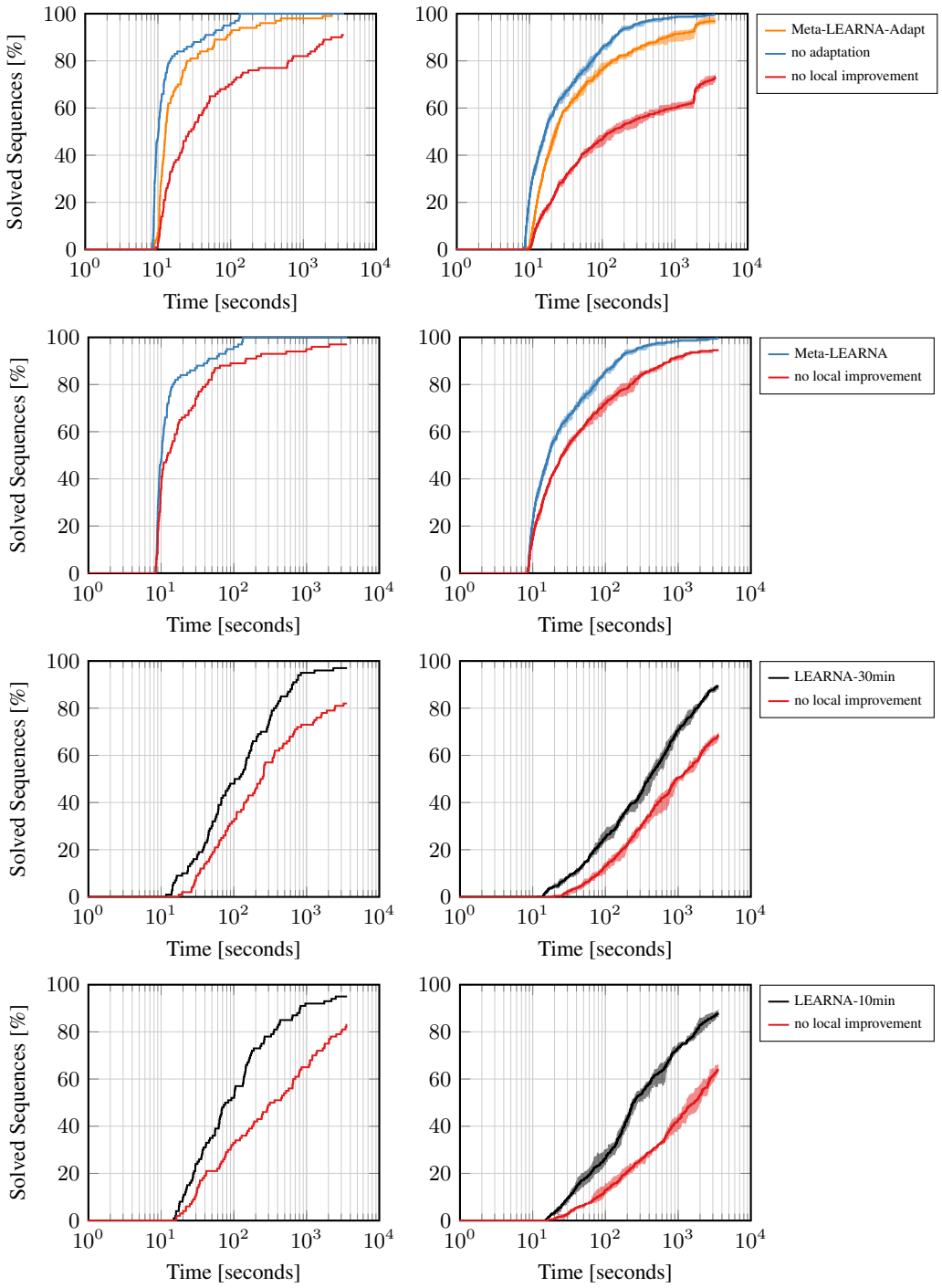


Figure 6: Ablation study of *Meta-LEARNA-Adapt* (first row), *Meta-LEARNA* (second row), *LEARNA-30min* (third row) and *LEARNA-10min* (fourth row) on Rfam-Learn-Test. The left side shows the accumulated number of solved target structures over 5 independent runs, while the right side shows the mean with confident interval.