

Authors:

Frederic Runge

University of Freiburg

Department of Computer Science

Georges-Köhler-Allee 74

79110 Freiburg, Germany

[runget@cs.uni-freiburg.de](mailto:runget@cs.uni-freiburg.de)

# Flexible RNA Design with Partial Constraints

## Abstract

RNA design is increasingly shaped by practical questions: which parts of an RNA must be preserved, which can be changed, and how much freedom a design method should have to explore alternatives. Traditional RNA design approaches often require designers to fully specify their goals upfront, even when only partial knowledge about sequence or structure is available. In this chapter, we introduce partial RNA design, a new design paradigm that separates task formulation from sequence generation. Rather than demanding complete and rigid specifications, partial RNA design allows designers to express what matters, such as conserved motifs, structural elements, or sequence regions, while leaving remaining degrees of freedom open for computational exploration. This enables intuitive, flexible, and reusable design formulations that closely mirror real-world RNA engineering workflows. We then present libLEARNNA as a practical tool that implements partial RNA design and generates diverse candidate RNA sequences consistent with the specified constraints. The chapter focuses on how partial RNA design tasks can be formulated and solved using libLEARNNA, guiding readers from conceptual design ideas to concrete RNA sequence candidates.

## 1 Introduction

RNA design describes the problem of finding an RNA sequence that fulfills a desired function. In practice, however, designing functional RNAs is rarely a matter of starting from scratch. More often, designers begin with partial information, such as a known secondary structure element, a conserved sequence motif, or a functional region that must remain unchanged. At the same time, large parts of the RNA may be deliberately left open to exploration. The central challenge is therefore not only to find RNA sequences, but to formulate design tasks in a way that preserves flexibility while still enabling computational guidance.

To address this challenge, we introduce partial RNA design [1], a design paradigm that treats RNA design as an interaction between two complementary components. First, a human defined formulation captures what should be fixed, what should be constrained, and what may vary. Second, a sequence generating algorithm fills in the remaining degrees of freedom in a principled and efficient manner.

A key property of partial RNA design is that constraints do not need to be enforced through a single mechanism. Structural or sequence constraints can be imposed directly through the design space, incorporated into the optimization objective, or provided as fixed context during sequence generation. This flexibility allows the same design formulation to be paired with different optimization goals.

An important consequence of this separation is that the same underlying sequence generator can be reused across multiple design objectives. Once a model has learned how RNA sequence patterns relate to structural features, it can be guided toward different downstream goals by modifying the objective function or evaluation criterion. For example, one design task may prioritize folding into a particular structural pattern, while another may emphasize covariation signals across sequence positions. Sequence and structure constraints can remain unchanged, be partially relaxed, or be reformulated as part of the objective itself.

From the user perspective, this means that changing what is optimized does not require redefining how RNA sequences are generated. This way of thinking supports exploratory design workflows in which objectives evolve over time. Designers can start with a simple formulation, inspect candidate sequences, and iteratively refine the objective to encourage additional properties without discarding previously defined constraints or redesigning the entire pipeline.

In the following, we first detail the concept of partial RNA design and how design tasks can be formulated for exploratory sequence generation. We then introduce libLEARN as a

practical generator that operationalizes this paradigm, before turning to a more detailed explanation of its algorithmic foundations.

## 1.1 Partial RNA Design: A User-Centric Paradigm

Partial RNA design formalizes the idea that an RNA design task does not describe a single target molecule, but rather a space of admissible solutions. This space is defined by combining fixed elements, constrained regions, and unconstrained regions, each reflecting a different degree of certainty about the desired RNA. The goal of partial RNA design is not to eliminate flexibility, but to control it in a way that aligns with the designer intent.

Fixed elements capture aspects that must be preserved, such as conserved motifs or essential sequence segments. Constrained regions specify properties that should be respected, for example a preferred structural context. Unconstrained regions are left open to exploration, allowing variability in sequence, structure, or length. Together, these elements form a flexible design formulation that reflects the designer intent while maintaining room for computational exploration.

A simple example is the design of RNA sequences of variable length that contain a GNRA tetraloop motif. In this case, the GNRA sequence itself is fixed, while the surrounding regions are left open. The tetraloop may be embedded in a preferred hairpin structure, but depending on the design objective, this structural context can be enforced strictly, encouraged, or simply provided as prior information. This formulation does not describe a single RNA or even a single structure, but rather a family of RNAs that share a common motif while allowing variability elsewhere.

A key aspect of partial RNA design is that constraints and objectives are conceptually decoupled. Sequence constraints are typically enforced directly by restricting which positions are designed, whereas structural constraints may play different roles depending on the optimization objective. When optimizing for structural consistency, structural constraints act

as hard requirements. In contrast, when optimizing for alternative objectives, such as generating sequences that match a particular RNA family, structural information may act as a guiding bias rather than a strict constraint, shaping the generation process without guaranteeing structural preservation in every candidate.

This distinction is intentional and reflects the flexibility of partial RNA design. By allowing constraints to function either as hard requirements or as contextual guidance, the same design formulation can support different exploratory goals. Changing what is optimized does not require redefining the design space. Instead, it changes how strongly different aspects of the formulation influence the generated sequences.

Because partial RNA design defines a space of possible solutions rather than a single target, its natural outcome is a set of diverse candidate RNAs. This aligns well with experimental workflows, where multiple designs are evaluated in parallel and where constraints may be gradually tightened or relaxed over time. Partial RNA design therefore shifts the role of computational methods from producing a single optimal RNA to supporting iterative and exploratory design processes.

In the following, we describe how partial RNA design formulations are instantiated in practice and how these ideas are realized by the libLEARNNA framework.

## 1.2 libLEARNNA

Partial RNA design requires more than a method that produces a single optimized sequence. Instead, it calls for a sequence generating approach that can adapt to different design formulations and objectives, and that can continue to improve while exploring a design space. libLEARNNA was developed with this perspective in mind.

At a high level, libLEARNNA can be understood as a system that learns how to design RNA rather than directly producing a single design. Its output is not a single RNA sequence, but a

learned sequence generation strategy that enables the software to produce many diverse candidate RNA sequences for the same design task.

The foundations of libLEARNNA lie in the earlier LEARNNA approach [2], which introduced automated reinforcement learning for RNA design. In LEARNNA, RNA sequence construction is modeled as a sequential decision process. Starting from an incomplete sequence, the algorithm decides which nucleotide to place at each position. What distinguishes LEARNNA from traditional reinforcement learning approaches is its level of automation. Rather than fixing the decision process, network architecture, or hyperparameters manually, these components are optimized jointly. The learning process therefore produces not only trained parameters, but an entire algorithmic design strategy that is well suited to RNA design.

libLEARNNA builds on this idea and extends it substantially. It is based on Meta LEARNNA Adapt [2], an approach in which the sequence generating algorithm continues to learn during sequence generation, guided directly by the design objective. While the model is initially trained on classical RNA design tasks related to inverse folding, it is not limited to this setting. Instead, it adapts its behavior online as it encounters new objectives, allowing it to remain effective even when the optimization goal is changed.

This capability is crucial for partial RNA design. In libLEARNNA, sequence and structural information are integrated into a unified task formulation. Sequence constraints are enforced directly by restricting which positions are designed, ensuring that fixed motifs or sequence regions are preserved. Structural information is incorporated during training and generation in a more flexible manner. When structural consistency is part of the objective (as during training), it is optimized explicitly. When alternative objectives are used, structural information influences sequence generation implicitly and acts as a learned prior rather than a hard requirement.

As a result, libLEARNNA is not limited to classical inverse RNA folding. Because it is trained on masked sequences and partially specified structures, it learns to operate under

incomplete information from the outset. This enables it to handle partial constraints, variable length designs, and objectives that go beyond strict structure matching. At the same time, its strong performance on inverse folding tasks provides a robust foundation that transfers naturally to other design objectives.

From a user perspective, libLEARNNA behaves like a fully automated design engine. The user specifies a partial RNA design formulation and an objective, and libLEARNNA generates diverse candidate sequences while continuously adapting its generation strategy to the task at hand. Changing the objective does not require retraining the model or redefining the algorithm. Instead, the learned design strategy adapts during generation.

In the remainder of this chapter, we focus on how libLEARNNA can be used in practice for partial RNA design. Using the design of variable length RNAs containing a GNRA tetraloop motif as a running example, we demonstrate how different objectives can be explored within the same formulation and how libLEARNNA supports iterative and exploratory RNA design workflows.

## 2 Materials

### 2.1 Installation and Software Requirements

libLEARNNA is available as an open-source project on github at [https://github.com/automl/learnna\\_tools/tree/libLEARNNA\\_reproduce](https://github.com/automl/learnna_tools/tree/libLEARNNA_reproduce), including detailed documentation and install guidelines. The installation guidelines include the setup of a conda environment using Python 3.6.

Besides libLEARNNA, the repository also provides command line interfaces to the LEARNNA, Meta-LEARNNA, and Meta-LEARNNA-Adapt algorithms [2, 3] that libLEARNNA is based on after installation. All algorithms can be executed on a single CPU of a standard notebook or desktop computer. The source code was tested on Linux.

## 3 Methods

This section guides the reader through the usage of libLEARNNA. It describes the scope of the program, its inputs and outputs, and presents example use cases to illustrate its behavior as an exploratory tool for RNA design.

### 3.1 Description of the Scope of the Program

libLEARNNA is an automated deep reinforcement learning algorithm based on the Meta LEARNNA Adapt approach [2]. The model is pretrained to learn a relationship between RNA sequence and structure and continues to learn from its own predictions for each task based on a quality signal provided by a design objective. This enables libLEARNNA to adapt to new design objectives quickly without retraining the model from scratch.

An integrated sampling procedure further allows libLEARNNA to design RNAs of different lengths within a single program execution. Together with its adaptive learning strategy, this enables libLEARNNA to explore large design spaces while improving its generation process over time. As a result, libLEARNNA is particularly well suited for generating libraries of many RNAs that share specified properties (see Note 1).

libLEARNNA natively supports structure-based RNA design using minimum free energy (MFE) or maximum expected accuracy (MEA) structure guidance from ViennaRNA's RNAFold [4], including designs with a desired GC content. In addition, it supports the generation of RNA sequences that optimize RNA-RNA interaction energies using IntaRNA [5], as well as sequence generation guided by covariance models of RNA families from the Rfam database [6]. In the following sections, we focus on the usage of libLEARNNA for structure-based RNA design to illustrate the principles of partial RNA design (see Note 2).

## 3.2 Input Formats

Sequence and structure constraints are provided to libLEARNNA in an input file using a predefined set of tags. The first line of the file starts with the character '>' followed by the name of the target, similar to the common FASTA file format. The target name is used during result storage and optional plotting. The second line specifies the sequence constraints and must start with the '#seq' tag. libLEARNNA supports the use of IUPAC nomenclature for RNA nucleotides to further increase design flexibility and exploration of the design space.

The sequence constraints are followed by a line of structural constraints that must start with the '#str' tag. libLEARNNA supports structural constraints provided in an extended version of the dot bracket notation [7], where a dot represents a non-interacting nucleotide and a pair of brackets represents a pair of interacting nucleotides. Currently, libLEARNNA supports RNA design for nested structures only (see Note 3).

A key aspect of libLEARNNA is the ability to specify partial sequence and structure constraints. For this purpose, libLEARNNA interprets the character 'N' as a wildcard symbol representing an unconstrained position in the sequence and or structure definition. To enable exploration of design spaces that include sequences of different lengths, libLEARNNA further allows the specification of points of possible extension. These positions indicate sites where a sequence may be extended or not and are marked by the character 'X'. The algorithm can then explore larger design spaces up to a user defined maximum overall sequence length. If multiple extension points are present, the maximum sequence length acts as a global constraint for the entire RNA, and libLEARNNA explores multiple solutions with different extension lengths at the positions marked with 'X' (see Note 4).

libLEARNNA also supports the definition of multiple targets within a single input file. These targets are processed sequentially within a single program execution. By default, libLEARNNA continues learning across all targets provided in the input file. The command line flag --

`no_shared_agent` can be used to disable this behavior and restart the learning procedure for each target individually (see Note 5).

Using our running example of RNAs that contain a GNRA tetraloop motif, a minimal input file for libLEARNNA could look as follows:

```
>Simple variable length GNRA Tetraloop example  
  
#seq XNNGNRANNX  
  
#str X((....))X
```

In this example, the character 'X' marks two positions of possible extension that flank the GNRA loop motif. At least two paired positions before and after the loop region are defined by the bracket notation, while the nucleotide composition of these paired regions is left unconstrained using the character 'N'. We will use this example file as input for the designs in the following sections to illustrate the usage and capabilities of libLEARNNA in more detail. For reproducibility, we recommend saving this example to a file named

```
gnra_tetraloop_design.input.
```

### 3.3 Output Format

libLEARNNA prints a tabular per-target summary of all solutions found to the terminal during execution. In addition, all results are saved to a directory on disk. By default, this directory is named `results`, but an alternative output directory can be specified using the `--results_dir <path>` command line option.

libLEARNNA supports multiple output formats through the `--output_format` parameter.

Supported formats include a FASTA-like format using the option 'fasta', a comma separated values format using the option 'csv', and a pickle-based format using the option 'pickle'. The FASTA-like format contains the target name, the designed sequence, and the corresponding secondary structure after folding with RNAFold. The csv and pickle formats store the results

in tabular form and provide additional information such as runtime, GC content, and sequence length.

We recommend using the csv format, as it contains the complete set of information produced during the design process and can be easily shared and analyzed across different platforms.

### 3.4 Basic Program Options

After successful installation, the `liblearna` command is available systemwide. A list of all command line options can then be obtained by running:

```
liblearna -h
```

We will now discuss some of the general parameters that can be set via the command line interface of libLEARNNA.

The input file is specified using the `--input_file <path>` option.

Since libLEARNNA allows to explore a design space rather than providing a single solution, the number of candidates that satisfy the provided constraints can be set via the `--num_solutions <natural number>` parameter. Designs for RNAs with different lengths further require setting a minimum and maximum length of the designed RNAs via the `--min_length` and `--max_length` parameters.

A minimal call to libLEARNNA for the design of RNAs between 20 and 100 nucleotides that contain a GNRA tetraloop would then look as follows:

```
liblearna --input_file <path to input file> --num_solutions <natural number> --min_length 20 --max_length 100
```

In the following, we detail the usage of libLEARNNA on explicit examples, introducing further command line parameters where necessary.

### 3.5 Exploring Large Design Spaces with libLEARNNA

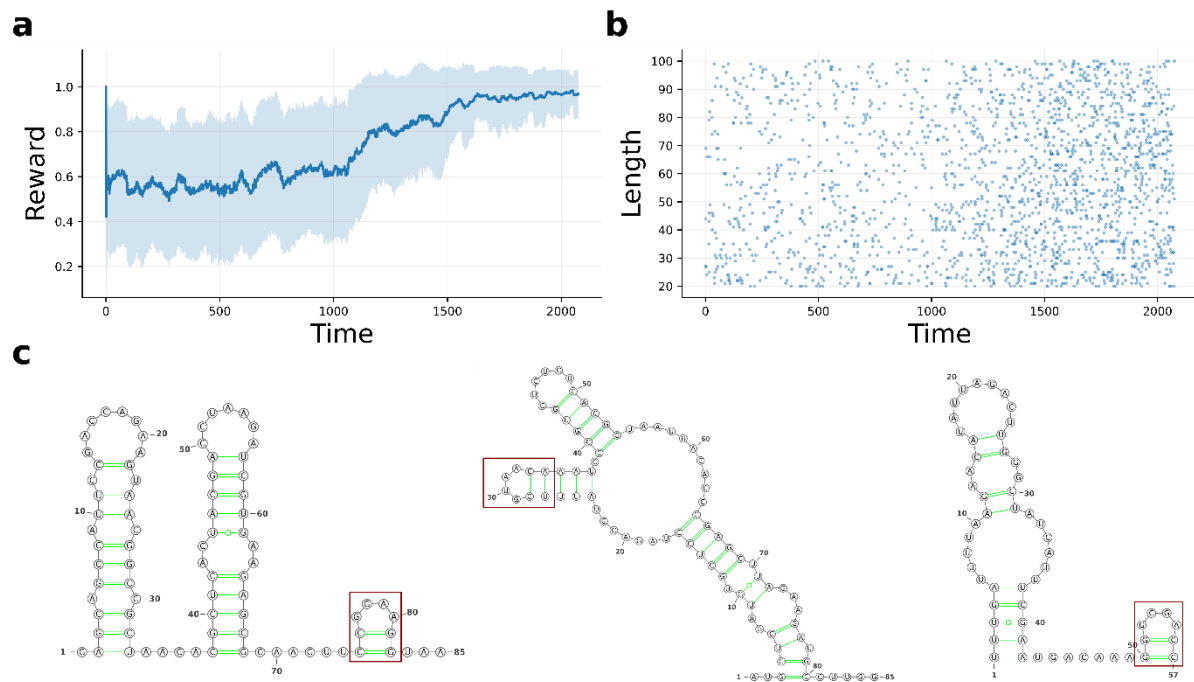
libLEARNNA was developed to support the exploration of large RNA design spaces and to provide designers with promising candidate RNAs for subsequent analysis. The design of RNAs that contain a GNRA tetraloop motif provides an example of such a design space with many possible solutions. It combines largely unrestricted regions, a well-defined motif with sequence and structural constraints, and the possibility of RNAs with different lengths.

To illustrate the exploration capabilities of libLEARNNA, we consider the GNRA design space and aim to generate a total of 2000 RNAs with lengths between 20 and 100 nucleotides that contain a GNRA tetraloop motif. The corresponding libLEARNNA call is as follows:

```
liblearnna --input_file gnra_tetraloop_design.input --num_solutions  
2000 --results_dir GNRA_results --output_format csv --min_length 20  
--max_length 100
```

Figure 1a shows the reward averaged over non-overlapping batches of 100 designs over time for this design task. The steadily increasing reward indicates that libLEARNNA successfully adapts to the task at hand. The total runtime is slightly above 2000 seconds, meaning that libLEARNNA produces a valid RNA design containing a GNRA tetraloop motif at a rate of approximately one sequence per second.

The resulting candidate RNAs are nearly uniformly distributed across the specified length range, as shown in Figure 1b, and exhibit substantial structural diversity while preserving the GNRA tetraloop motif at various sequence positions, as illustrated in Figure 1c.



**Figure 1: Exploration of large partial RNA design spaces using libLEARN.** (a) Batch-averaged reward over time with standard deviation, computed over batches of 100 designs, for the task of designing RNAs of length 20 to 100 nucleotides containing a GNRA tetraloop motif. The steadily increasing reward indicates libLEARN’s adaptation to the design task over time. (b) Scatter plot showing the length of successful candidate designs over time. Solutions are distributed across the full allowed length range, with a slight increase of successful designs at later stages, consistent with increasing reward values. (c) Representative example RNA secondary structures generated by libLEARN, illustrating diversity in sequence length and global structure while preserving the GNRA tetraloop motif.

### 3.6 Design-Space Exploration Under Varying Objectives

A central feature of partial RNA design is the separation of the task definition from the actual design goal or objective. To demonstrate the benefit of this separation, we apply libLEARN to the same task definition for RNAs containing GNRA tetraloops while varying the design objective.

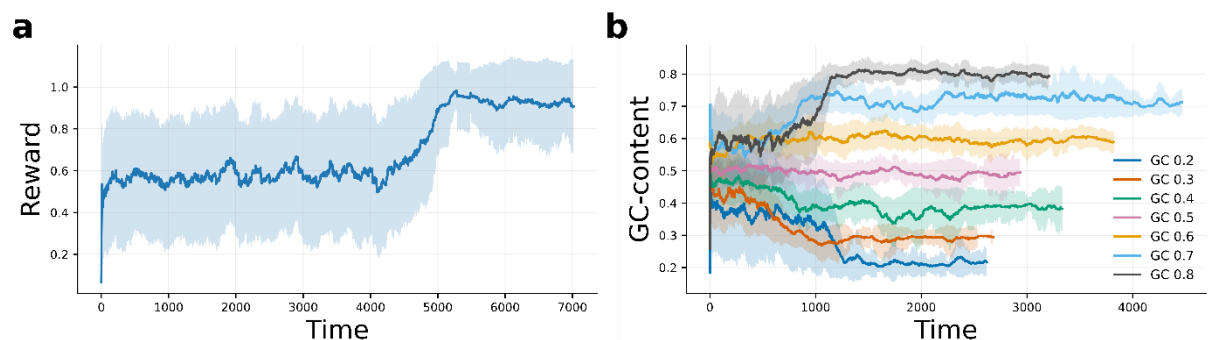
#### 3.6.1 Switching the Folding Objective

As described previously, libLEARN uses RNAFold as its default folding algorithm to learn relationships between RNA sequence and structure during training. In this setting, structural information is obtained by folding sequences into their minimum free-energy (MFE) structures. RNAFold also supports folding based on a maximum expected accuracy (MEA) objective, which can result in different predicted structures compared to MFE-folding.

To assess libLEARNNA's ability to adapt to a changed objective, we switch RNAFold to MEA-folding and rerun libLEARNNA without modifying the underlying task definition of designing RNAs that contain GNRA tetraloops. The folding algorithm can be changed using the `--algorithm` command line parameter. The command used for this experiment is as follows:

```
liblearnna --input_file gnra_tetraloop_design.input --num_solutions
2000 --results_dir GNRA_results --output_format csv --min_length 20
--max_length 100 --algorithm rnafold_mea
```

Figure 2a shows the reward averaged over batches of 100 designs over time for the design using the MEA-folding objective. The increasing reward indicates that libLEARNNA successfully adapts to the new objective. The overall runtime is approximately three times longer than for designs using MFE-folding, which reflects the increased difficulty of the task. Nevertheless, libLEARNNA still produces a valid design approximately every 3.5 seconds on average, despite operating under conditions that differ from its original training setup.



**Figure 2: libLEARNNA adapts to changes in the design objective.** (a) Batch-averaged reward over time with standard deviation, computed over batches of 100 designs, for the design of GNRA loop RNAs using maximum expected accuracy (MEA) folding instead of minimum free energy (MFE) folding. The increasing reward indicates successful adaptation of libLEARNNA to the changed folding objective. (b) Batch-averaged GC-content with standard deviation, computed over batches of 100 designs, for RNA sequences generated by libLEARNNA over time for different target GC values (0.2 to 0.8). For each specified GC value, libLEARNNA adapts its sequence generation strategy to sample sequences close to the desired GC-content.

### 3.6.2 RNA Design with Desired GC-Content

In addition to structure-based exploration under different folding objectives, libLEARNNA supports RNA design with a specified GC-content. The desired GC-content can be defined per target in the input file using a dedicated '#GC'-tag or set globally using the `--`

`desired_gc <float>` command line parameter (see Note 6). By default, libLEARNNA allows a deviation of 0.01 from the specified GC-content. This tolerance can be adjusted using the `--gc_tolerance <float>` parameter.

For the GNRA tetraloop design example, a desired GC-content of 0.3 can be specified using the following command:

```
liblearnna --input_file gnra_tetraloop_design.input --num_solutions
2000 --results_dir GNRA_results --output_format csv --min_length 20
--max_length 100 --desired_gc 0.3
```

Importantly, libLEARNNA has not been trained to explicitly optimize GC-content. Instead, the GC-content objective is introduced as an additional penalty term that complements the structure-based objective and guides the sequence generation process (for details see [1]). To visualize libLEARNNA adaptation to this new objective, we repeat the GNRA tetraloop design task for different desired GC values.

As shown in Figure 2b, libLEARNNA adapts to the different GC-content objectives over time, increasingly sampling sequences that fall within the desired GC-content range. Compared to designs without GC-content constraints, the additional objective leads to increased runtime, similar to the change observed when switching the folding objective. However, even for the longest run at a desired GC-content of 0.7, libLEARNNA still produces a valid design approximately every 2.5 seconds on average.

### 3.7 RNA Design-Spaces as a Combination of Local Motifs

In the previous sections, we demonstrated libLEARNNA's ability to explore large design spaces and to adapt to different design objectives. The design example used so far, however, was relatively simple and contained only a single GNRA tetraloop motif. One of the most notable features of partial RNA design is the ability to combine multiple motifs within a single design task while still leaving room for computational exploration.

To demonstrate such a combined design, we consider RNA design tasks that contain multiple GNRA tetraloop motifs. The GNRA motif is defined by a sequence component 'NNGNRANN' and a corresponding structural component '((...))'. As an example, we combine three such motifs into a single task formulation, separated by potential points of extension, resulting in the following input definition:

```
>GNRA design multi

#seq XNNGNRANNXNNGNRANNXGNRANNX

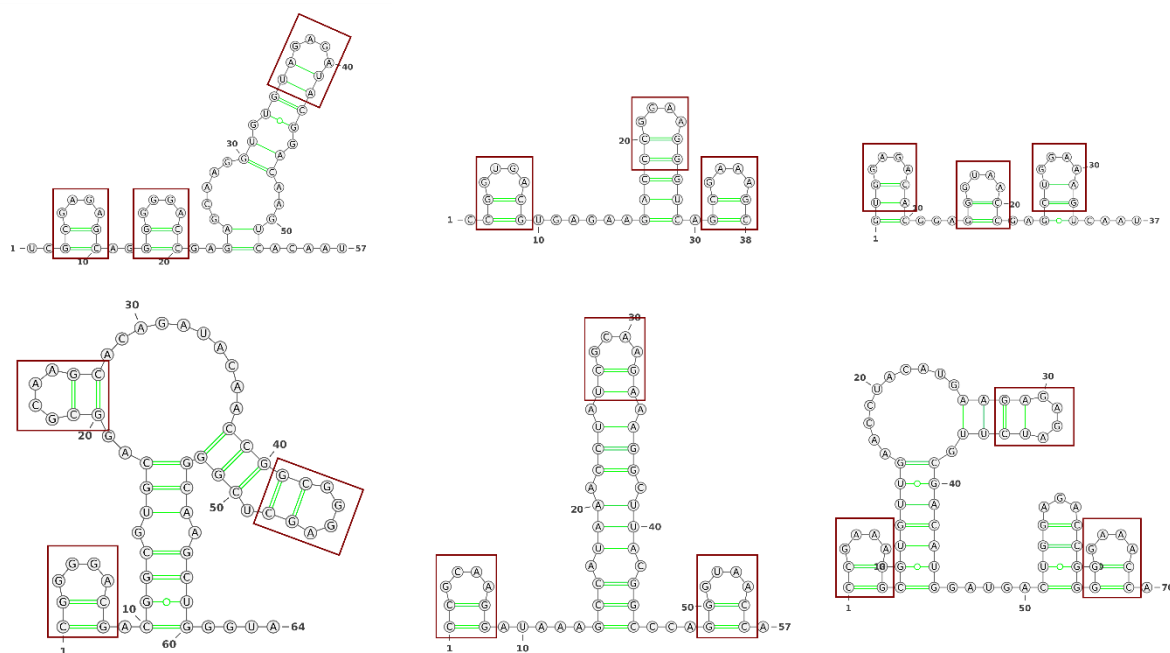
#str X(...))X(...))X(...))X
```

After saving this definition to a file named `gnra_tetraloop_multi_design.input`, `libLEARNa` can be executed to generate RNA designs that contain at least three GNRA tetraloop motifs. For this example, we generate a total of 20 candidate RNAs using the following command:

```
liblearn --input_file gnra_tetraloop_multi_design.input --
num_solutions 20 --results_dir results_gnra_design_multi --
output_format csv --min_length 20 --max_length 100 --plot_structure
```

To account for the increased number of fixed positions introduced by the multiple motifs, we increase the minimum sequence length from 20 to 30 nucleotides. Further, we use the `--plot_structure` parameter of `libLEARNa` to directly plot the structures using VARNA [8].

Examples of `libLEARNa` designs that contain at least three GNRA tetraloop motifs are shown in Figure 3. Due to the largely unrestricted nature of the design space, the resulting RNAs are highly diverse in terms of sequence length and structural features, while the three GNRA tetraloop motifs are preserved in every design. These results again highlight a key aspect of partial RNA design. In contrast to classical RNA design paradigms, partial RNA design tasks define a space of possible solutions rather than a single target. This flexibility, however, requires the designer to carefully formulate the design space to ensure that the generated solutions align with the intended design goals (see Note 7).



**Figure 3: Example solutions generated by libLEARN for the multi GNRA loop design task.**

Shown are RNA sequences of different lengths and structural compositions generated by libLEARN for a design task requiring the presence of three GNRA tetraloop motifs. Red boxes highlight the GNRA loops in each design. Despite substantial variability in global structure and sequence length, all examples satisfy the specified motif constraints.

## 4 Notes

Note 1 libLEARN continues learning during sequence generation for the specific task at hand. This enables the algorithm to adapt both to the defined design space and to new objectives specified by the designer. Due to the non-deterministic nature of the learning process, however, the algorithm may converge to a local optimum from which it cannot easily escape. To address this behavior, libLEARN provides the `--restart_timeout <seconds>` parameter. This option allows the user to specify a time interval after which all neural network weights are reset, effectively restarting the task specific optimization without requiring a manual restart of the program. Using a restart timeout can be particularly useful when generating a large number of sequences for a task that requires prolonged adaptation. Based on our experience, restarting libLEARN every 1800 seconds can often be beneficial, although the optimal setting depends on the specific design task.

Note 2 Besides structure-based RNA design, libLEARNNA provides built in reward functions for generating RNA sequences that maximize RNA-RNA interaction energies as well as for generating sequences that match a given Rfam covariance model. In this chapter, however, we focus on structure-based RNA design to explain the core ideas of partial RNA design. Additional details on the generation of sequences for these alternative objectives, including example commands and installation instructions, are available in the libLEARNNA repository at [https://github.com/automl/learnna\\_tools/tree/libLEARNNA\\_reproduce](https://github.com/automl/learnna_tools/tree/libLEARNNA_reproduce).

Note 3 libLEARNNA was developed for the design of nested RNA structures using structure information provided by ViennaRNA's RNAFold. In principle, it is possible to switch the folding algorithm used within libLEARNNA. While pseudoknotted structures are not natively supported, coupling libLEARNNA with a structure prediction method that handles pseudoknots and using an appropriate task formulation could already enable RNA design involving pseudoknotted regions. For example, pseudoknotted positions could be left unconstrained in the structure definition using the character 'N'. Native support for pseudoknotted structures, however, would require retraining the algorithm with an extended structure alphabet, which would substantially increase the state space and the complexity of the learning problem.

Note 4 The sampling procedure of libLEARNNA is based on inserting sequence segments of variable length at positions marked as points of extension using the character 'X'. Depending on the specified sequence length range and the number of extension points defined in the design space, not all positions marked with 'X' are necessarily extended in every iteration. For example, the sampled insertion at the first extension point may already reach the maximum allowed sequence length, in which case subsequent extension points are not considered in that iteration. In later iterations, however, the sampled insertion lengths differ, allowing libLEARNNA to explore a broad range of sequence lengths and structural compositions over time. This behavior is reflected in the distribution of sequence lengths shown in Figure 1b.

**Note 5** The `--no_shared_agent` flag in libLEARNNA can be useful for reducing the workload of the designer in certain scenarios. This option was introduced to distinguish between closely related design tasks and collections of unrelated tasks processed within a single program execution. For closely related tasks, it can be beneficial to share learning across tasks, allowing the algorithm to reuse information from successful designs in earlier tasks. In contrast, when multiple unrelated tasks are defined, continuous learning across tasks may negatively affect performance on later targets. In such cases, disabling shared learning using the `--no_shared_agent` flag ensures that each task is optimized independently.

**Note 6** Please note that a desired GC-content set via command line overrides any GC-values provided in the input file for individual tasks. Multiple different GC-contents for multiple tasks can only be defined per task definition in the input file.

**Note 7** libLEARNNA is capable of exploring arbitrarily large or even fully unrestricted RNA design spaces and can generate large sets of candidate solutions. However, the formulation of the design space is critical for achieving the intended outcomes. As an example, consider the design of RNAs containing multiple GNRA tetraloop motifs. While all resulting designs contain the specified number of GNRA loops, the exploratory nature of libLEARNNA combined with a largely unrestricted design space can lead to solutions that include additional loops, varying numbers of stems, or internal loop structures that may not be desired. Accurate and careful definition of the design space with respect to the intended outcome is therefore particularly important when using libLEARNNA. Compared to classical RNA design approaches that require a single fully specified target structure, partial RNA design places more responsibility on the designer to define meaningful constraints. When formulated appropriately, however, this flexibility enables exploration of rich design spaces that would be difficult to access with more rigid design paradigms.

## 5 Conclusion

In this chapter, we introduced partial RNA design as a user centric RNA design paradigm that supports flexible and expressive task formulations. By separating the specification of design constraints from the sequence generation process, partial RNA design enables exploratory workflows and design tasks that are difficult to address using traditional RNA design approaches.

We demonstrated how partial RNA design can be realized in practice using libLEARNNA, an automated reinforcement learning based implementation that adapts to different design objectives while operating on the same underlying task formulation. Using simple and illustrative examples, we showed how libLEARNNA can explore large design spaces, adapt to changing objectives, and generate diverse candidate RNA sequences that satisfy user defined constraints.

We expect partial RNA design to become increasingly relevant as RNA engineering tasks grow more modular, objective driven, and exploratory. Future implementations of this paradigm may further expand its applicability and contribute to more flexible and scalable RNA design methodologies.

## 6 References

1. Runge F, Franke JKH, Fertmann D et al. (2024) Partial RNA Design. *Bioinformatics* 40:i437-i445. <https://doi.org/10.1093/bioinformatics/btae222>
2. Runge F, Stoll, D., Falkner, S., Hutter F (2019) Learning to Design RNA. *International Conference on Learning Representations*
3. Runge F, Hutter F (2025) Machine Learning for RNA Design: LEARNNA. *Methods Mol Biol* 2847:63–93. [https://doi.org/10.1007/978-1-0716-4079-1\\_5](https://doi.org/10.1007/978-1-0716-4079-1_5)
4. Lorenz R, Bernhart SH, Höner zu Siederdisen C et al. (2011) ViennaRNA Package 2.0. *Algorithms for Molecular Biology*
5. Mann M, Wright PR, Backofen R (2017) IntaRNA 2.0: Enhanced and Customizable Prediction of RNA–RNA Interactions. *Nucleic Acids Research*:W435-W439
6. Oniveros-Palacios N, Cooke E, Nawrocki EP et al. (2025) Rfam 15: RNA families database in 2025. *Nucleic Acids Research*:D258-D267
7. Hofacker IL, Fontana W, Stadler PF et al. (1994) Fast folding and comparison of RNA secondary structures. *Monatshefte für Chemie* 125:167–188. <https://doi.org/10.1007/BF00818163>

This is an unpublished author manuscript that has been accepted for publication as a chapter in a Springer Methods series. This version has not undergone final copyediting and typesetting and may differ from the published version of record.

8. Darty K, Denise A, Ponty Y (2009) VARNA: Interactive drawing and editing of the RNA secondary structure. *Bioinformatics* 25:1974–1975.  
<https://doi.org/10.1093/bioinformatics/btp250>