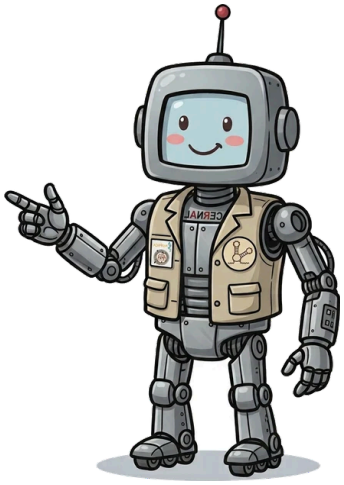
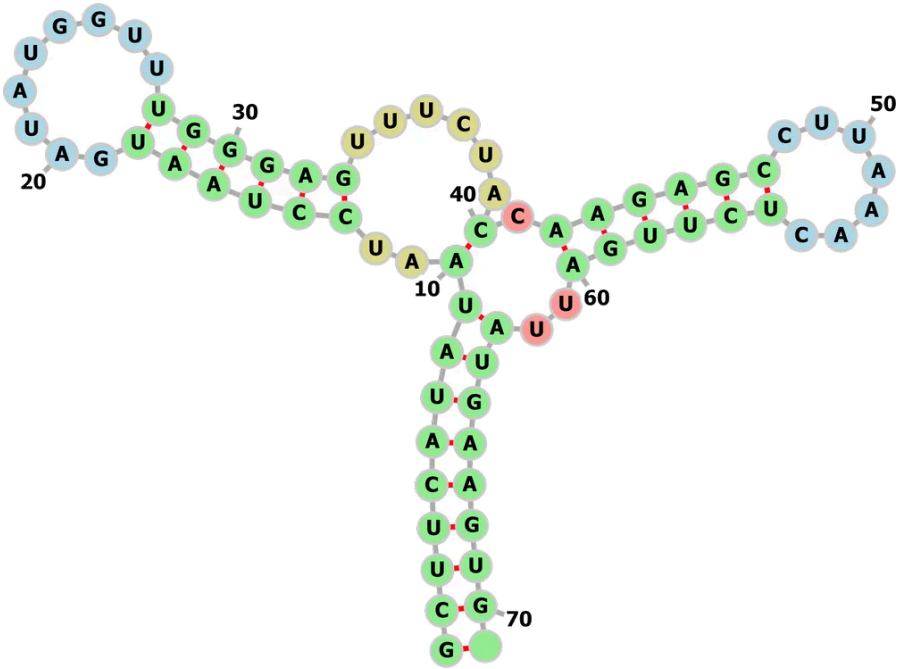
ViennaRNA & NUPACK *for Beginners*



RNA Folding from Scratch

by the CERNAL team

Everything you need to fold RNA with ViennaRNA and NUPACK - starting from what a base pair is, ending with designing your own switch. No prior computational background assumed.



This is what you're aiming for: a 73-nucleotide sequence, folded, drawn in forna.
The colours mark the parts you'll learn to name in chapter 2.

stems · hairpin loops · interior loops and bulges · the multiloop junction

15 chapters · ~90 min read · runnable code · live folding engine

Contents

PART ONE - WHAT YOU'RE ACTUALLY ASKING

01 Why RNA folds

02 Dot-bracket notation

03 Energy and ΔG

PART TWO - THE IDEA THAT TRIPS EVERYONE

04 MFE: the best structure

05 The ensemble

06 Base-pair probabilities

PART THREE - GETTING THE TOOLS RUNNING

07 ViennaRNA or NUPACK?

08 Installing ViennaRNA

09 Installing NUPACK + WSL

PART FOUR - DOING REAL WORK

10 Your first fold

11 Two strands

12 Accessibility

13 From analysis to design

PART FIVE - THINGS THAT GO WRONG

14 Gotchas hall of fame

15 Where to go next

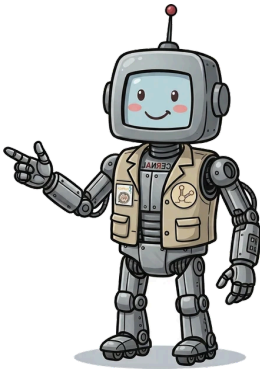
This PDF is the print edition. The four folding widgets are interactive in the web version: 2026.igem.wiki/tau-israel/rna-folding-guide

01 Why RNA folds, and why you should care

DNA is almost always double-stranded: two separate molecules, each one the mirror image of the other, zipped together. There is nothing left over to fold.

RNA is usually produced as a *single* strand. But it is still made of the same sticky letters - A, U, G, C - and those letters still want to pair. With no partner strand around, the only thing an RNA molecule can pair with is **itself**. It folds back and grabs its own tail.

Which parts grab which other parts is not random. It is determined by the sequence, and it can be predicted. That prediction is what this entire guide is about.



Your guide for the next 15 chapters.

The pairing rules

Only certain letters stick to each other, and some stick harder than others:

Pair	Hydrogen bonds	Strength	Name
G–C	3	Strongest	Watson–Crick
A–U	2	Medium	Watson–Crick
G–U	2 (offset)	Weak	Wobble

> **The one everybody forgets**
>
> `G–U` is real. It is weaker than the other two and it does not appear in the pairing rules you learned in first-year biology, but RNA forms it constantly and every folding program counts it. If you ever write your own check for "are these two sequences complementary?", and you only check G–C and A–U, you will quietly miss a third of the pairing that actually happens.

Why does it matter for your project?

Because folding decides whether a sequence *works*. A ribosome cannot start translating a messenger RNA if the start region is buried inside a hairpin. A trigger RNA cannot bind to your switch if the binding site is already paired up with something else nearby.

So the practical questions you will be asking a folding program are almost always one of these three:

1. **What shape does my sequence take?** - give me a structure.
2. **Is this particular region open or buried?** - give me accessibility.
3. **Will these two sequences stick to each other?** - give me binding energy.

ViennaRNA and NUPACK are two different pieces of software that answer those questions. Everything else in this guide is detail.

What these tools do *not* do

Worth knowing up front, so you don't waste a week:

- They predict **secondary structure** - which base is paired to which base. They do not give you a 3D shape with coordinates in space.
- They mostly cannot predict **pseudoknots** (we'll explain these in chapter 2). Real RNA forms them; standard algorithms skip them.
- They model a molecule sitting alone in a tube of salt water at equilibrium. A real cell has proteins, ribosomes, crowding, and RNA that folds while it is still being made. Predictions are a very good starting hypothesis, not the truth.

> **Set your expectations correctly**
>
> These programs are excellent at *comparing* things - this variant folds better than that one, this region is more open than that one. They are much less reliable at absolute claims - "this exact structure is what the molecule looks like in a cell." Design your work around comparisons and you will rarely be badly wrong.

02 Dot-bracket: the notation everything speaks

Before any tool will talk to you, you have to learn its handwriting. Fortunately it has exactly three characters.

A folded RNA is just a list of which positions are paired to which. You could write that as a table, but it would be unreadable. Instead everyone uses **dot-bracket notation**: one character per nucleotide, lined up underneath the sequence.

- `.` - this base is unpaired.
- `(` - this base is paired with a base *later* in the sequence.
- `)` - this base is paired with a base *earlier* in the sequence.

Brackets match the way they do in arithmetic: the innermost `(` goes with the nearest `)`. A valid structure always has equal numbers of each.

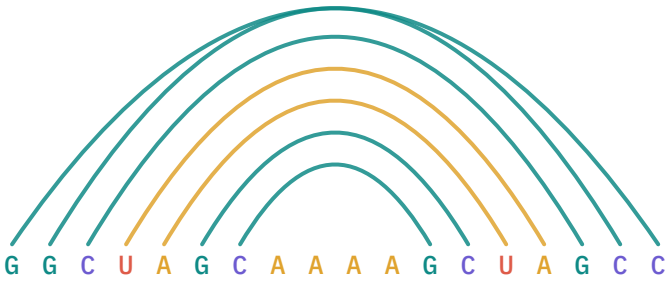
Read it both ways Live in the web version: 2026.igem.wiki/tau-israel/rna-folding-guide

Sequence

```
GGCUAGCAAAAGCUAGCC
```

Structure

```
(((((((....)))))))
```



7 base pairs. Valid structure.

A U G C

The vocabulary of shapes

Once you can read dot-bracket, you can name what you see. These five words cover almost everything you'll meet:

Name	Looks like	What it is
Stem (helix)	`((((  ))))`	A run of consecutive pairs. The stable part - this is what holds a structure together.
Hairpin loop	`((((....))))`	Unpaired bases closed off by a single stem. Needs at least 3 unpaired bases - RNA physically can't bend tighter than that.
Bulge	`(((.(((...))))))`	Unpaired bases on *one* side of a stem only.
Internal loop	`(((..` `(((...)))..)))`	Unpaired bases on *both* sides.
Multiloop	`((..((..))..` `((..))..))`	A junction where three or more stems meet.

Pseudoknots, and why you won't see them

A pseudoknot is when a loop pairs with something *outside* its own enclosing stem - in bracket terms, `((((....[[[[....))))....]]]]`, where the brackets cross instead of nesting.

Real RNA does this. Standard folding algorithms do not predict it, because allowing crossing pairs makes the problem enormously harder to compute. Both ViennaRNA and NUPACK will happily hand you a pseudoknot-free answer for a molecule that in reality forms a pseudoknot, and nothing in the output will warn you. If your sequence is a known riboswitch or ribozyme, go look up its real structure in the literature rather than trusting a prediction.

Counting from zero, counting from one

This sounds trivial. It will cost you a day at some point, so read it now.

Python counts from 0. Biologists count from 1. The folding libraries are inconsistent with each other about which convention they use in which function. When you write `seq[10:20]` and compare it to a structure coordinate that a tool reported as "position 10", you have a decent chance of being off by one and never noticing, because an off-by-one answer still looks completely plausible.

How to not lose that day

Pick a convention for your own code, write it in a comment at the top of the file, and then *print the actual slice* every time you extract a region. If you expect `AUG` and you print `UGA`, you just saved yourself eight hours.

03 Energy: what ΔG actually means

Every number these tools give you is in kcal/mol, and more negative always means more stable. If you remember nothing else from this chapter, remember that.

When two bases pair, the molecule releases a little energy and settles into a more comfortable state. When a loop forms, the molecule has to be bent and held in place, which *costs* energy. The free energy ΔG of a structure is the sum of all those gains and costs.

- **Negative ΔG** - the folded structure is more stable than the unfolded chain. The molecule wants to be in this shape.
- **ΔG near zero** - barely holding together. It will fall apart and reform constantly.
- **Positive ΔG** - the structure costs more than it gains. It won't form on its own.

Why both tools give slightly different answers

ViennaRNA and NUPACK both estimate energies from experimentally measured tables (the Turner rules), but not from the same version. ViennaRNA defaults to the Turner 2004 set, while NUPACK uses its own `rna06` set, built from the same lineage with additional terms. They can also score some loop types differently. Folding the same sequence in both can give somewhat different ΔG values, and occasionally a genuinely different structure.

This is normal and is not a bug in either one. It does mean that **you must never mix numbers from the two tools in the same comparison.** Score all your candidates with one tool, then rank.

Temperature, salt, and the settings you'll forget

Base pairing is temperature-dependent - heat shakes pairs apart. It is also salt-dependent, because the RNA backbone is negatively charged and needs positive ions nearby to let two strands approach each other. Magnesium especially.

Both tools default to **37 °C**, which is right for mammalian and *E. coli* work and wrong for almost everything else. Salt defaults differ between the two. If you're designing something for a 30 °C incubator or a room-temperature cell-free reaction, set it explicitly.

Write your conditions down once, use them everywhere

Put temperature and salt in variables at the top of your script and pass them to every single call. The most common silent error in a folding pipeline is one function being called at the default 37 °C while everything else runs at your real temperature.

The quantity you'll actually use: RT

One constant shows up everywhere in the next three chapters. At 37 °C:

```
RT ≈ 0.616 kcal/mol
```

It is the typical amount of thermal energy knocking molecules around. It sets the scale for "is this energy difference big or small?" A structure that is 0.3 kcal/mol better than another is, thermally speaking, barely better at all. A structure 3 kcal/mol better is dominant. Everything about ensembles comes down to comparing energy differences to RT.

Part two

The idea that trips everyone up

If you only read three chapters of this guide, read these. The difference between *the* structure and *the ensemble* is the single most common source of wrong conclusions in RNA design.

04 MFE: the single best structure

Out of every shape your sequence could possibly fold into, one has the lowest energy. That one is the MFE structure.

MFE stands for **minimum free energy**. When you run the simplest command in either tool and it hands you back a dot-bracket string and a number, that is the MFE structure and its ΔG.

The clever part is that the program does not try every possible structure one at a time - there are astronomically many. It uses dynamic programming: it solves the best structure for every tiny sub-piece of the sequence, then builds those answers up into the best structure for the whole thing. The runtime scales as the cube of the length, which is why folding 100 nucleotides is instant and folding 10,000 is not.

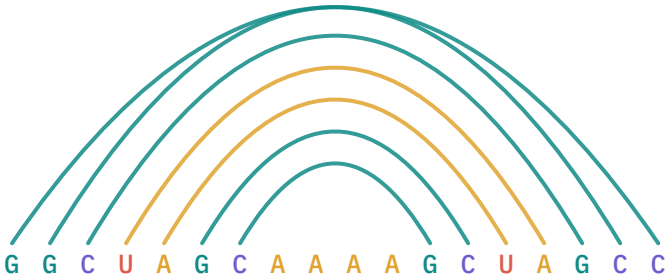
Fold something yourself

The widget below is a real folding engine running in your browser. It *exhaustively enumerates every single possible structure* of your sequence and finds the lowest-energy one - the brute-force approach that real tools are an optimization of.

The energy model is a simplified version of the real one. Watson–Crick stacks use the actual Turner 2004 values from the Nearest Neighbor Database (Turner & Mathews, 2010; Mittal et al., 2024), including their measured enthalpies, so stability changes with temperature the way it does in the real model. Three things are simplified: any stack involving a G–U pair uses the average A–U stack; closing a helix with a loop costs a flat +4.5 kcal/mol at every temperature, in place of the real size-dependent loop terms; and end penalties, dangling ends and terminal mismatches are left out. That is why short helices struggle to form here, but also why the numbers won't match ViennaRNA. The behaviour is the thing to watch.

Brute-force folder Live in the web version: 2026.igem.wiki/tau-israel/rna-folding-guide

```
GGCUAGCAAAAGCUAGCC
```




```
GGCUAGCAAAAGCUAGCC
((((((((....))))))))
```

MFE	structures checked	share of the MFE shape	time
-11.09 kcal/mol	736	99.1%	4 ms

The MFE structure is probably not what your molecule looks like

This is the trap. The MFE is the single best shape out of billions - but "best" might mean it beats the runner-up by 0.2 kcal/mol, which is less than a third of RT. In that case the molecule spends nearly as much time in the runner-up shape as in the "winner".

A dot-bracket string looks so definite that it is almost impossible not to believe it. Treat it as "the most likely single answer in a field of close contenders" until you have checked how close the contenders are. Chapter 5 shows you how.

When MFE alone is enough

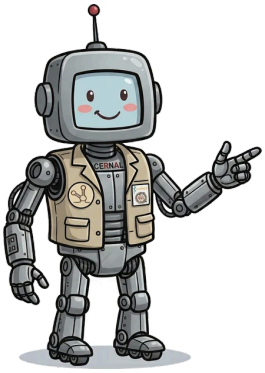
It isn't useless - it's just not the whole story. MFE on its own is fine when:

- You're sanity-checking that a designed hairpin forms at all.
- You're comparing many variants and only need a rough ranking.
- The structure is strongly determined - a long perfect stem with no alternatives.

It is **not** enough when you're asking whether a specific region is accessible, how reliable a design is, or whether two conformations compete. Those are ensemble questions.

05

The ensemble: why one structure is a lie



Not one shape. A crowd of them, and the crowd is the thing you actually measure.

A test tube holds trillions of copies of your molecule. They are not all in the same shape.

Each individual molecule is wiggling, breaking pairs, forming new ones. At any instant, the population is spread across many different structures. That whole population - every possible structure, each with a share of the molecules - is called the **thermodynamic ensemble**.

How big is each share? Boltzmann's law:

$$P(\text{structure}) = e^{-\Delta G / RT} / Q$$

Lower energy means a bigger share, and the relationship is exponential. `Q` - the **partition function** - is just the sum of `e^(−ΔG/RT)` over every possible structure. It's the normalizer that makes all the shares add up to 1.

Because RT ≈ 0.616 kcal/mol at 37 °C, the arithmetic works out to a rule of thumb worth memorizing:

Energy gap vs. the MFE	How much rarer that structure is
0.6 kcal/mol	about 2.7× rarer
1.4 kcal/mol	about 10× rarer
2.8 kcal/mol	about 100× rarer
4.3 kcal/mol	about 1000× rarer

So a structure 1 kcal/mol worse than the MFE is still a serious presence in the tube. A structure 5 kcal/mol worse is irrelevant.

Watch the crowd shift

Same brute-force engine, now showing the whole population rather than just the winner. Drag the temperature slider and watch the shares move.

The ensemble, live Live in the web version: 2026.igem.wiki/tau-israel/rna-folding-guide

GCGCAAAAGCGCAAAAGCGC

Temperature 37 °C

#	structure	ΔG	share
1	........((((....))))	-4.7	33.1%
2	((((....)))).........	-4.7	33.1%
3	((((............))))	-4.7	33.1%
4	.........(((....))).	-1.3	0.1%
5	........(((......)))	-1.3	0.1%
6	.(((....)))..........	-1.3	0.1%
7	.(((............))).	-1.3	0.1%
8	(((......))).........	-1.3	0.1%

Top 8 structures shown - together 99.8% of the population. The remaining 367 structures share the rest.

MFE	G ensemble	gap	completely unfolded
-4.70	-5.38	0.68	0.0%

structures
375

The sequence loaded by default is deliberately ambiguous. The same four G–C pairs can form in three different places (two small hairpins, or one stem with a long loop), and all three have exactly the same energy, so no structure wins: each holds a third of the population. Now switch the sequence to the clean hairpin `GGCUAGCAAAAGCUAGCC` and the picture changes completely: one structure takes about 99% of the population. **That difference is the whole point.** Two dot-bracket strings that look equally confident describe two completely different physical situations.

Then push the temperature up. Pairs weaken as they warm, the gaps between structures shrink, and the share of completely unfolded chains climbs: watch the "completely unfolded" number above 70 °C. That is melting, and it's why you have to tell these tools what temperature you're working at.

Ensemble free energy

Both tools can report a second energy number, computed from the partition function:

$$G_{ensemble} = -RT \cdot \ln(Q)$$

It summarizes the stability of the *whole population*, not one structure. It is always slightly lower (more negative) than the MFE, because it accounts for all the alternative shapes contributing as well.

> **The gap between them is a free diagnostic**
>
> Compare `MFE` with `G_ensemble`. If they're nearly equal, one structure dominates and your molecule is well-behaved. If the ensemble energy is much lower, the population is spread over many competing shapes and any single dot-bracket picture you show in a presentation is misleading.
>
> For a designed switch you generally *want* that gap small - a switch that can't decide what shape it is will not switch reliably.

06 Base-pair probabilities and the BPP matrix

Instead of asking "what is the structure?", ask "for each pair of positions, what fraction of molecules have them paired?" That answer is far more honest, and it is the output you should be using for most real work.

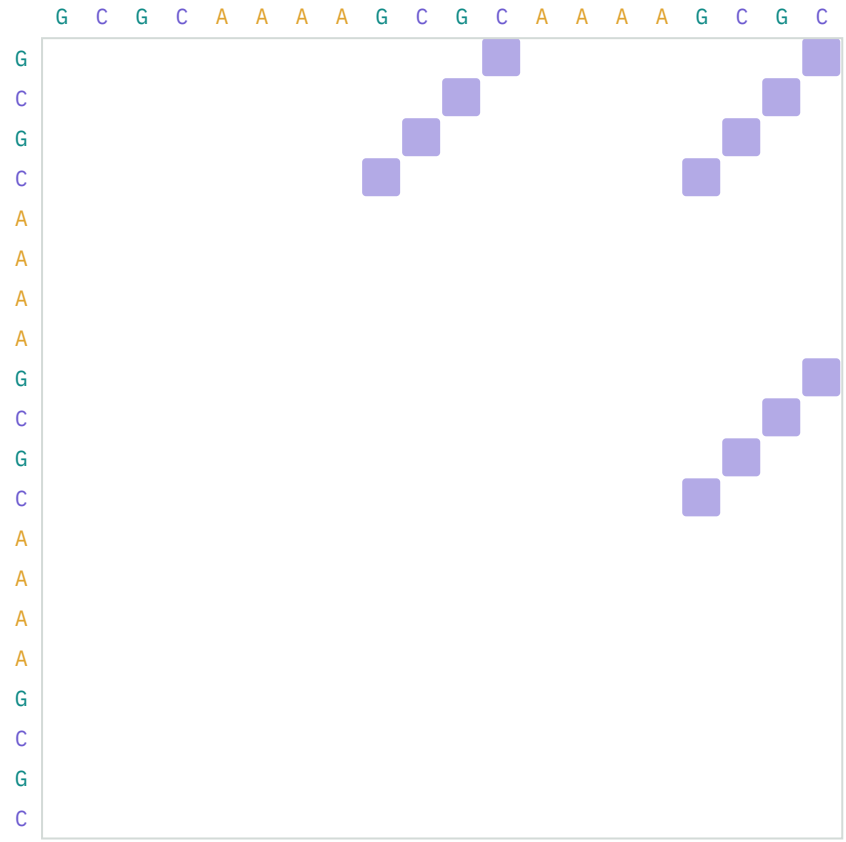
Add up the Boltzmann share of every structure in which position *i* is paired to position *j*. You get a number between 0 and 1: the **base-pair probability**. Do it for every pair of positions and you have the **BPP matrix**.

Displayed as a heatmap, it tells you in one glance what a dot-bracket string can never tell you: which parts of the structure are certain, and which are a coin flip.

Base-pair probability matrix

Live in the web version: 2026.igem.wiki/tau-israel/rna-folding-guide

```
GCGCAAAAGCGCAAAAGCGC
```



Most accessible base: A at position 5 (unpaired 100% of the time). Hover any cell for its pair probability.

Dark, sharp diagonals mean a confident helix. Faint, smeared blocks mean the molecule is torn between alternative registers - and that is exactly the situation where the MFE structure would have lied to you with a clean, confident-looking string of brackets.

Unpaired probability - the one you'll use most

For design work, the question is usually not "is *i* paired to *j*" but "is *i* paired to **anything**". Sum row *i* of the matrix over all partners, and subtract from 1:

$$P(i \text{ unpaired}) = 1 - \Sigma_j P(i,j)$$

This single number answers almost every accessibility question you have. Is my ribosome binding site exposed? Is my toehold free to meet its trigger? Average `P(unpaired)` over the region and you have a score you can rank designs by. We'll compute it in both tools in chapter 12.

ViennaRNA's matrix is upper-triangular and 1-indexed

When you call `fc.bpp()` in ViennaRNA you get a matrix where only `bpp[i][j]` with `i < j` is filled in. The lower triangle is zeros, and the indices start at 1, not 0 - row 0 and column 0 are padding.

If you loop over it naively and sum only `bpp[i][j]` for a fixed `i`, you will miss every partner that comes *before* position `i`, and silently compute unpaired probabilities that are far too high. Sum both `bpp[i][j]` and `bpp[j][i]`, and offset your indices by one.

NUPACK's pair matrix has the opposite shape of surprise: it is 0-indexed, fully filled, and has an *extra final column* holding the unpaired probability directly. Convenient - but if you take its width as the sequence length, every index you compute will be wrong.

Part three

Getting the tools onto your machine

The least intellectually interesting part of this guide, and the part most likely to make you give up. Budget an afternoon and don't take it personally.

07 ViennaRNA or NUPACK?

They overlap a lot. The short version: ViennaRNA for one strand, NUPACK for several strands at once.

	ViennaRNA	NUPACK
Best at	Folding a single strand; scanning long sequences; accessibility profiles	Mixtures of several strands; what actually forms at given concentrations; sequence design
Multiple strands	Two at a time, with some effort	Native. The whole design of the tool.
Concentrations	No	Yes - give it 1 µM of each strand and it tells you what you get
Speed	Very fast	Slower, especially with many strands
Install	One pip command	Register, download, install from a local file
Licence	Free to use under its own licence: research, educational and commercial use are allowed, with credit to the authors. Contact them to include it in a commercial product.	Free for non-commercial academic use after free registration. Commercial use needs a separate licence.
Command line	Excellent - `RNAfold`, `RNAplfold`, `RNAduplex` and more	Minimal; it's a Python library

How most people end up using both

A very common and sensible division of labour, and the one we'd recommend:

- **ViennaRNA for the cheap pass.** You have 700 nucleotides of target transcript and you need to score hundreds of candidate windows. Vienna is fast enough to do that in seconds.

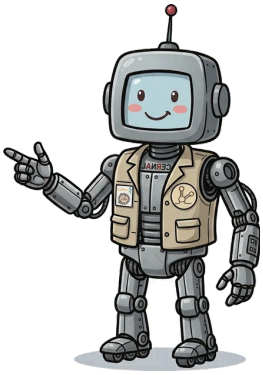
- **NUPACK for the expensive verdict.** You have 15 surviving candidates and you need to know what really happens when the switch and the trigger are in the same tube at realistic concentrations. That's a question only NUPACK answers properly.

This is also how CERNAL, our team's switch-design platform, works: it scores many candidate regions quickly with ViennaRNA, then passes the best ones to NUPACK for the final ranking.

> **If you only install one**
>
> Install ViennaRNA. It takes one minute, it answers most questions, and you can learn everything in part two of this guide with it. Add NUPACK when you have a genuine multi-strand question - which, if you're building a switch that binds a trigger, you will.

08 Installing ViennaRNA

Make a virtual environment first. Always. It costs ten seconds and saves you from the situation where fixing one project breaks another.



If this chapter goes badly, it is not you. It is the package manager.

```
any operating system - make a clean environment

# create a folder for your project and go into it
mkdir rna-work && cd rna-work

# create the environment
python3 -m venv .venv

# activate it - macOS / Linux / WSL
source .venv/bin/activate

# activate it - Windows PowerShell
.venv\Scripts\Activate.ps1
```

Your prompt should now start with `(.venv)`. If it doesn't, the activate step didn't work, and nothing after this will behave.

Then install it

```
install ViennaRNA

pip install ViennaRNA
```

That's genuinely the whole thing on macOS, Linux and WSL. Check it worked:

```
check

python -c "import RNA; print(RNA.fold('GGCUAGCAAAAGCUAGCC'))"
```

You should see a dot-bracket string and a negative number. If you do, you're done.

The three systems

System	What to do
macOS	The pip command above works. If you don't have `python3`, install it from python.org or with `brew install python`.
Linux	The pip command works. If `python3 -m venv` complains, install the venv package first: `sudo apt install python3-venv python3-pip`.
Windows	Try the pip command directly in PowerShell first - it often just works now. If it fails with a compiler error, don't fight it: set up WSL (next chapter) and treat it as a Linux machine from then on.

The command-line programs

The pip package gives you the Python library. It does *not* always give you the standalone commands `RNAfold`, `RNAplfold` and friends. If you want those too - and they're handy for quick one-off checks - install via conda instead:

```
conda, if you want the command-line tools as well

conda install -c bioconda viennarna
```

Then this works straight from your terminal, no Python needed:

```
fold a sequence without writing a script

echo "GGCUAGCAAAAGCUAGCC" | RNAfold
```

09 Installing NUPACK (and surviving WSL)

NUPACK is not on PyPI. You cannot `pip install nupack` from the internet. This confuses everyone exactly once.

Because of its licence, you have to register and download it yourself, then install from the file you downloaded.

Step by step

1. Go to **nupack.org** and create a free account. Non-commercial academic use is free after you register; commercial use needs a separate licence.
2. Download the NUPACK 4 source archive from the downloads page.
3. Unzip it somewhere you can find again.
4. Activate your virtual environment (see the previous chapter).
5. Install from the unzipped folder.

```
install from the folder you unzipped - adjust the version number

# the -f flag points pip at a local folder instead of the internet
pip install -U nupack -f ./nupack-4.0.1.12/package
```

Check it:

```
check

python -c "from nupack import Model; print(Model(material='rna', celsius=37))"
```

> **NUPACK 4 does not run natively on Windows**
>
> There is no Windows build. If you are on Windows, you need WSL - a real Linux running inside your Windows machine. This is not a workaround or a hack; it is the supported path, and once it's set up it's genuinely pleasant.

Setting up WSL, the short version

Open PowerShell *as administrator* and run:

```
PowerShell, as administrator

wsl --install
```

Reboot when it asks. You'll get an Ubuntu terminal and be prompted to pick a username and password - write the password down, you'll need it for `sudo`. Then, inside that Ubuntu terminal:

```
inside Ubuntu, first time only

sudo apt update
sudo apt install python3-pip python3-venv unzip
```

From here, follow the macOS/Linux instructions exactly. WSL *is* Linux.

Three things that make WSL much less painful

Keep your files on the Linux side. Work in `/home/yourname/project`, not in `/mnt/c/Users/...`. Crossing between the two filesystems is dramatically slower - slow enough that you'll think your script has hung when it's just reading files.

Use VS Code's WSL extension. Install it, then type `code .` inside your Ubuntu terminal. You get the normal Windows editor, editing Linux files, with a Linux terminal built in. No file copying.

Get to your Linux files from Windows Explorer by typing `\\wsl$` in the address bar, when you need to drag something in or out.

When your WSL session dies mid-run

If WSL freezes or disconnects while a folding script is running, the instinct is to blame NUPACK. Usually it's memory - WSL takes a fixed share of your RAM, and a loop that builds a very large list of candidates will exhaust it and get the whole session killed.

Before you start debugging the science, put a counter in your loop and print how many candidates you're actually generating. A combinatorial expansion that you expected to produce a few thousand variants can quietly produce a few hundred million.

Part four

Doing real work

From here on, everything is code you can paste and run. Every block appears twice - once in ViennaRNA, once in NUPACK - so you can see the same idea in both dialects.

10 Your first fold, in both tools

One sequence, one strand, and the four numbers you'll want from it: the MFE structure, its energy, the ensemble free energy, and the per-base unpaired probability.

ViennaRNA

vienna_first_fold.py

```
import RNA

seq = "GGCUAGCAAAAGCUAGCC"

# set your conditions explicitly - never rely on defaults
md = RNA.md()
md.temperature = 37.0          # degrees Celsius

fc = RNA.fold_compound(seq, md)

# 1. the MFE structure and its energy
structure, mfe = fc.mfe()
print(seq)
print(structure, f"{mfe:.2f} kcal/mol")

# 2. the ensemble - rescale first, or you risk numerical overflow
fc.exp_params_rescale(mfe)
pf_structure, g_ensemble = fc.pf()
print(f"ensemble free energy: {g_ensemble:.2f} kcal/mol")
print(f"gap to MFE:           {mfe - g_ensemble:.2f} kcal/mol")
```

Call `exp_params_rescale` before `pf()`

The partition function multiplies together a very large number of exponentials. For long sequences this overflows the floating-point range and you get nonsense or an error. Passing the MFE to `exp_params_rescale` first tells the library what scale to expect. On an 18-nucleotide toy it makes no difference; on a 500-nucleotide transcript it is the difference between a result and a crash.

NUPACK

nupack_first_fold.py

```
from nupack import Model, Strand, Complex, complex_analysis

seq = "GGCUAGCAAAAGCUAGCC"

model = Model(material='rna', celsius=37,
              sodium=1.0, magnesium=0.0)

a  = Strand(seq, name='a')
cx = Complex([a], name='a-alone')

result = complex_analysis([cx], model=model,
                          compute=['pfunc', 'mfe', 'pairs'])
r = result[cx]

print(r.mfe[0].structure, f"{r.mfe[0].energy:.2f} kcal/mol")
print(f"ensemble free energy: {r.free_energy:.2f} kcal/mol")
```

Two things to notice about NUPACK's style. First, **everything flows through a `Model`** - temperature and salt live there and get passed to every call, which makes it much harder to accidentally mix conditions. Second, **you name your objects**. With one strand that's pointless; with six strands in a tube, those names are what keeps the output readable.

Unpaired probability, in both

ViennaRNA - note the index handling

```
bpp = fc.bpp()           # 1-indexed, UPPER TRIANGLE ONLY
n = len(seq)

unpaired = []
for i in range(1, n + 1):
    paired = 0.0
    for j in range(1, n + 1):
        if i < j:
            paired += bpp[i][j]
        elif j < i:
            paired += bpp[j][i]     # <-- the half everyone forgets
    unpaired.append(1.0 - paired)

for i, (base, p) in enumerate(zip(seq, unpaired)):
    print(f"{i:3d} {base}  {p:.3f}")
```

```
NUPACK - it hands you the column directly

P = r.pairs.to_array()    # shape is (n, n+1), NOT (n, n)
unpaired = P[:, -1]        # last column = probability of being unpaired

for i, (base, p) in enumerate(zip(seq, unpaired)):
    print(f"{i:3d} {base}  {p:.3f}")
```

> **Sanity-check your first run against the picture**
>
> Paste your sequence and structure into **forna** (a free structure viewer that runs in the browser) and look at it. If the hairpin loop in the drawing isn't where your unpaired probabilities are highest, you have an indexing bug. This takes thirty seconds and catches an enormous fraction of mistakes.

11 Two strands: binding, complexes, concentrations

This is where the two tools genuinely diverge, and where NUPACK starts to earn its installation hassle.

You have a switch and a trigger. You want to know whether they bind, how strongly, and - the question people forget to ask - whether they'd rather do something else instead, like bind to themselves.

The quick answer: ViennaRNA

```
fast duplex check

import RNA

switch  = "GGCUAGCAAAAGCUAGCC"
trigger = "GGCUAGCUUUUGCUAGCC"

# duplexfold: only intermolecular pairs, very fast, good for screening
duplex = RNA.duplexfold(switch, trigger)
print(duplex.structure, f"{duplex.energy:.2f} kcal/mol")

# cofold: allows BOTH intra- and intermolecular pairing - slower, more honest
fc = RNA.fold_compound(switch + "&" + trigger)
ss, dimer_mfe = fc.mfe_dimer()
print(ss, f"{dimer_mfe:.2f} kcal/mol")
```

`duplexfold` ignores the competition

`duplexfold` only considers pairs *between* the two strands. It will cheerfully report a beautiful −25 kcal/mol duplex for two sequences that, in reality, are both locked up in their own hairpins and never meet.

Use it for fast screening of thousands of candidates. Never use it for a final verdict. The cost of actually opening up each strand's own structure is a real energy cost, and it is the reason most designs that look great on paper don't work.

The honest answer: NUPACK

NUPACK's native unit is a **tube**: strands at given concentrations, and it computes the equilibrium of every complex they could form.

what is actually in the tube at equilibrium

```
from nupack import Model, Strand, Tube, SetSpec, tube_analysis

model = Model(material='rna', celsius=37)

sw = Strand("GGCUAGCAAAAGCUAGCC", name='switch')
tg = Strand("GGCUAGCUUUUGCUAGCC", name='trigger')

# 1 micromolar of each; consider every complex of up to 2 strands
t1 = Tube(strands={sw: 1e-6, tg: 1e-6},
          complexes=SetSpec(max_size=2),
          name='t1')

result = tube_analysis(tubes=[t1], model=model, compute=['pairs'])

for cx, conc in result.tubes[t1].complex_concentrations.items():
    print(f"{cx.name:20s} {conc:.3e} M")
```

The output is a list of every species - switch alone, trigger alone, switch+trigger, switch+switch, trigger+trigger - with how much of each exists at equilibrium. That is the question you actually care about, and no amount of ΔG arithmetic substitutes for it.

Homodimers are the silent killer

Set `max_size=2` and read the whole output, not just the row you were hoping for. A switch that binds its trigger beautifully but binds *a second copy of itself* even better will never work, and a two-strand ΔG calculation will never tell you. This single check has saved more designs than any other thing in this guide.

Binding energy, done properly

The energy of forming the complex is not the energy of the complex. It's the difference between the complex and the two separated strands, each folded as it would be on its own:

$$\Delta G_{binding} = \Delta G(AB) - \Delta G(A) - \Delta G(B)$$

```
binding energy from ensemble free energies
```

```python
from nupack import Complex, complex_analysis

cx_sw  = Complex([sw],     name='sw')
cx_tg  = Complex([tg],     name='tg')
cx_bth = Complex([sw, tg], name='sw+tg')

res = complex_analysis([cx_sw, cx_tg, cx_bth], model=model, compute=['pfunc'])

dG_bind = (res[cx_bth].free_energy
           - res[cx_sw].free_energy
           - res[cx_tg].free_energy)

print(f"binding dG: {dG_bind:.2f} kcal/mol")
```

Use `free_energy` (the ensemble value) rather than the MFE here. You're comparing populations, not single structures, and the ensemble value is the one that reflects what the molecules are actually doing.

12 Accessibility: is this region open?

The most common real question in RNA design, and the one with the most ways to get it subtly wrong.

"Is my ribosome binding site exposed?" "Can the trigger reach this window?" Both reduce to: average the unpaired probability over a region.

accessibility of a region, ViennaRNA

```
def accessibility(seq, start, end, temperature=37.0):
    """Mean unpaired probability over seq[start:end]. 0-indexed, end exclusive."""
    md = RNA.md(); md.temperature = temperature
    fc = RNA.fold_compound(seq, md)
    _, mfe = fc.mfe()
    fc.exp_params_rescale(mfe)
    fc.pf()
    bpp = fc.bpp()
    n = len(seq)

    total = 0.0
    for i0 in range(start, end):
        i = i0 + 1                    # bpp is 1-indexed
        paired = sum(bpp[i][j] for j in range(i + 1, n + 1)) \
               + sum(bpp[j][i] for j in range(1, i))
        total += 1.0 - paired
    return total / (end - start)
```

> **Context changes the answer completely**
>
> Folding a 30-nucleotide window on its own and calling it accessible is meaningless. In the cell that window is embedded in a 1,500-nucleotide transcript, and the rest of the transcript is very likely pairing with it.
>
> Always fold the window **inside its real flanking context**. And since you don't know how much context matters, check several flank sizes - 0, 25, 50, 100 nucleotides on each side - and take the *worst* result, not the best. A region that looks open only when you crop it tightly is not open.

Long sequences: use a sliding window

Folding a whole 3,000-nucleotide mRNA is slow and, worse, physically questionable - long-range pairs predicted across thousands of nucleotides are rarely real. ViennaRNA's `RNAplfold` handles this properly by only considering pairs within a limited span:

RNAplfold - accessibility profile along a long transcript

```
# -W 150 : only consider pairs within a 150nt window
# -u 20  : report probability that each 20nt stretch is fully unpaired
RNAplfold -W 150 -u 20 < transcript.fa
```

This gives you a profile along the whole transcript, and it's the right tool for "where in this gene should I target?"

13 From analysis to design

So far you've been asking "what does this sequence do?" Design flips it: "what sequence would do this?"

This is called **inverse folding**. You specify the structure you want and let the software search sequence space for something that folds into it.

ViennaRNA: one structure, quick and simple

```
inverse folding
```

```
import RNA

target = "(((((((....)))))))"
start  = RNA.random_string(len(target), "ACGU")

seq, distance = RNA.inverse_fold(start, target)
print(seq, f"(distance {distance})")

# distance 0 means it folds exactly into the target
print(RNA.fold(seq))
```

Fast, and fine when you need "a hairpin, any hairpin". Its limitation is that it optimizes for one structure in isolation - it has no concept of a second strand or a competing state.

NUPACK: multiple states at once

This is what NUPACK was built for, and it's the right tool for a switch. You describe the states you want - the switch open when alone, the switch closed when the trigger is present - and NUPACK searches for a sequence that satisfies *all* of them simultaneously.

```
sketch - check your version's docs for exact argument names

from nupack import *

model = Model(material='rna', celsius=37)

# N = any base; the designer picks these
stem = Domain('N7',  name='stem')
loop = Domain('N4',  name='loop')

s = TargetStrand([stem, loop, ~stem], name='s')   # ~ = reverse complement

c = TargetComplex([s], "(((((((....)))))))", name='hairpin')

t = TargetTube(on_targets={c: 1e-6},
               off_targets=SetSpec(max_size=2),
               name='t')

job = tube_design(tubes=[t], model=model)
best = job.run(trials=1)[0]
print(best)
```

Two ideas there are worth absorbing even if you never run the code:

- **Domains with constraints.** `N7` means seven free bases. You can also freeze sequence you aren't allowed to change - a ribosome binding site, a start codon, a sequence you must match to an endogenous target.
- **Off-targets.** `SetSpec(max_size=2)` tells the designer to actively penalize unwanted complexes, including the strand binding itself. The design isn't just "folds right" - it's "folds right *and* doesn't do anything else."

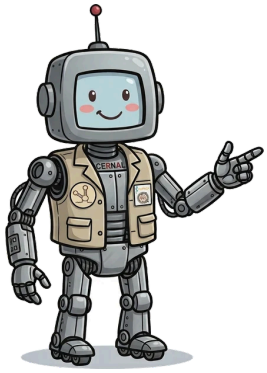
> **Design gives you candidates, not answers**
>
> A design run that reports a low defect score has found a sequence satisfying the constraints *you wrote down*. If your constraints were incomplete - you forgot to forbid an internal start codon, you didn't screen against the host transcriptome - the result will be beautifully optimized and biologically useless.
>
> Always run your designed sequences back through the analysis workflow in chapters 10–12 as an independent check, and screen them against the real background they'll live in.

Part five

Things that will go wrong

Every item below has cost a real person real days. None of them produce an error message - that's what makes them expensive.

14 The gotchas hall of fame



Nine ways to be confidently, silently wrong.

1. T instead of U

You pulled a sequence from a genome file, so it's DNA: `ATGC`. You're folding it as RNA. Depending on the tool and the version, a `T` may be silently converted, silently ignored, or treated as an unpairable character - and each of those gives a different wrong answer.

Fix: convert explicitly and assert, every time you read a sequence. `seq = seq.upper().replace("T", "U")`, then `assert set(seq) <= set("ACGU")`.

2. Folding DNA with the RNA energy model

NUPACK's `Model(material='rna')` and `Model(material='dna')` use entirely different parameter sets. If you're designing a DNA oligo and you left it on `'rna'`, your energies are wrong by several kcal/mol and nothing will tell you.

3. The upper-triangular, 1-indexed matrix

Covered in chapter 6, repeated here because it is the single most common silent bug in this whole field. ViennaRNA's `bpp()` fills only `i < j` and starts at index 1. Sum both triangles. NUPACK's matrix is 0-indexed with an extra column on the end. Print the shape before you index into it.

4. A transcription leader shifting all your coordinates

You add a few nucleotides to the front of your construct - a `GGG` for efficient T7 transcription, a cloning scar, a restriction site. Every coordinate you had carefully worked out now points three positions too early.

The insidious part is that the checks still *pass*. Your script reports the start codon is accessible; it is actually reporting on the two bases before it and the first base of it. **Fix:** never hard-code positions. Find features by searching for them in the final assembled sequence, and print what you found.

5. Checking complementarity without G–U

Writing your own "are these complementary?" check with only G–C and A–U misses wobble pairs. In a screen for off-target binding, that means you pass sequences that in reality bind quite well.

6. Comparing numbers from different tools

ViennaRNA and NUPACK energies are not interchangeable, and neither are their ensemble free energies. Rank within one tool. If you must use both, use one for screening and the other for the final verdict - never mix them inside the same comparison.

7. Forgetting that an energy threshold depends on length

"We required ΔG below −20 kcal/mol." For a 200-nucleotide construct, that's weak. For a 25-nucleotide one, it's impossible. Any absolute energy cutoff applied across candidates of different lengths will systematically select the long ones. Normalize by length, or compare only like with like.

8. Combinatorial explosion

"Try all synonymous codings of this peptide" sounds bounded. A 15-amino-acid peptide has billions of synonymous encodings. Your script will consume all available memory and the process - or the whole WSL session - will be killed with no useful message.

Fix: before any exhaustive enumeration, *count* first. If the count exceeds a limit you set, sample randomly instead of enumerating.

9. Believing a threshold that nothing can pass

You set a quality gate - say, 90% of the binding region must be paired - and run thousands of candidates. Nothing passes. The instinct is to keep searching harder.

Stop and ask whether the threshold is physically reachable. If parts of your region are fixed sequence that cannot pair with the target by construction, the maximum achievable score may be well below your gate, and no amount of searching will help. Compute the ceiling before you set the floor.

> **The habit that prevents most of these**
>
> After every step in your pipeline, print the actual sequence of the actual region you think you're working on. Not the coordinates - the letters. Almost every bug on this list announces itself immediately if you're looking at the letters.

15 Where to go next

You now know enough to be dangerous. Here's where to go to become reliable.

Look at your structures

- **forna** - paste a sequence and dot-bracket, get a clean interactive drawing. Browser-based, nothing to install. Use it constantly.
- **The ViennaRNA web server** - RNAfold, RNAplfold and others with no installation, good for a quick check or showing a colleague.
- **NUPACK's web app** - analysis and design in the browser, free with your academic account.

A sensible first project

If you want to consolidate all of this, build one small pipeline end to end:

1. Take a gene of interest and fold it with a sliding window to find the most accessible 30-nucleotide region.
2. Design an antisense sequence against that region.
3. Check in NUPACK that the two bind, and that neither forms homodimers.
4. Check that the antisense sequence doesn't bind strongly to anything else in the host transcriptome.

That is, in miniature, what every RNA design project does. Everything else is refinement.

When you are ready to design a switch

You are welcome to use **CERNAL**, our team's software, to see how these tools come together into a logic circuit tailored to your goal and to the cell of your choice.

> **Now go and fold something**
> Good luck.

ViennaRNA & NUPACK for Beginners - RNA Folding from Scratch, by the CERNAL team. A community contribution. Free to reuse, adapt and translate for any iGEM team or classroom; attribution appreciated.

Fonts: Fraunces, Public Sans and IBM Plex Mono, embedded under the SIL Open Font License 1.1.