Supplementary Methods and Notes

Joint optimisation of amino acid and coding sequence for de novo designed proteins

S1. Lattice construction and k-best extraction

The lattice

For a backbone of length L, ProteinMPNN is run once in unconditional mode (--unconditional_probs_only) to obtain a matrix of per-position log-marginals over the twenty amino acids. At each position i, the set of admitted amino acids is

$$A_i(\delta) = \{a : \log p_i(a) \geq \max_b \log p_i(b) - \delta\}$$

where δ is the entropy budget in nats. Setting δ to zero collapses each position to its marginal argmax, and the lattice reduces to a fixed-protein codon lattice. The states at position i are the sense codons for every admitted amino acid, so the lattice is degenerate in both amino acid and codon. An optional anchor sequence adds its residue at every position to $A_i(\delta)$ regardless of the budget. This option is used wherever a comparison must contain a specified starting sequence, such as the design's own sequence in the cross-host study.

Node and edge weights

The node weight at position i for state c is

$$w(i, c) = \lambda_{mpnn} \log p_i(aa(c)) + \lambda_{cai} \log a_h(c) + \lambda_{gc}\, gc(c)$$

where a_h is the relative adaptiveness in host h and gc(c) is the GC fraction of the codon. The weight of an edge between adjacent states is

$$w(i, c \rightarrow c') = \lambda_{cpb}\, cps_h(c, c')$$

where cps is the codon-pair score of Coleman et al. The objective is the sum of the node weights and the edge weights along a path. Every term is additive over nodes or over adjacent pairs, and it is this property that makes exact optimisation possible.

k-best paths

A single Viterbi pass recovers the optimum. The k-best list is built from per-state prefix lists: at each position and state, the best k prefixes reaching that state are retained in a list formed by a k-way merge over the incoming states using a heap. The total work is $O(L S^2 k \log S)$ for S

states per position, where S is at most 6 for a fixed-protein lattice and at most 62 for an unbounded shell. A 43-residue lattice at $\delta = 1.0$ is solved in a median time of 1.6 ms.

The list is exact for the objective it is given: it is the true top-k under the sum of node and edge weights, with no beam search and no pruning.

What the lattice cannot carry

Two classes of terms are excluded by construction. The mRNA folding free energy is not a sum over positions, because base pairs form at an arbitrary range. Protein-level liabilities are functions of the whole residue sequence. Both are therefore evaluated on finished candidates in the rescoring tier (see Methods, Two-tier architecture).

S2. Term normalisation

The problem

The Tier-1 score is expressed in nats, whereas the initiation-window folding energy is expressed in kcal/mol. Adding the two directly treats the weight as an implicit unit conversion, so a weight fitted to one backbone does not transfer to another backbone of a different length.

Measured conversion

Across the candidate pools, the median spreads are 6.317 nats for the Tier-1 score and 2.376 kcal/mol for the initiation term, a ratio of 2.67 nats per kcal/mol. An unnormalised weight of 1.0 therefore weighted the folding term at 0.37 of parity, and the unnormalised grid of 0.1, 0.3 and 1.0 corresponds to 0.36, 1.09 and 3.63 on the normalised scale.

Normalisation

Each term is divided by its standard deviation over the candidate pool being ranked, so that the weights are dimensionless:

$$\text{total} = \text{tier1} / s_{\text{tier1}} + w_{\text{init}} \times \text{initiation} / s_{\text{init}} - \Sigma_j \, w_j \times \text{liability}_j / s_j$$

Scales are measured once per backbone on the pool to be ranked. Where several arms are compared on one backbone, they share a single pool, so the scales do not shift between arms.

A term whose spread over the pool is numerically zero is divided out rather than floored. A term that takes the same value on every candidate cannot order them, and a small floor would allow it to dominate every other term by orders of magnitude.

Reparameterisation test

The test tests/test_mrna.py asserts that, with unit scales, the ranking reproduces the unnormalised run, and that normalisation changes the weight at which a given ordering occurs without changing the ordering itself at the corresponding weight.

S3. Baseline invocation

Two of the baselines required configuration that is not documented upstream.

CodonMPNN

The weights are published at

https://publbuck.s3.us-east-2.amazonaws.com/codonmpnn_afdb_taxons20k.ckpt (SHA-256 prefix 00b0070ac77bf1f5) and the code at github.com/HannesStark/CodonMPNN, at commit 71e81cf. The repository provides no inference entry point; likelihood_eval.py uses absolute paths to the authors' cluster and evaluates the likelihood for a single dataset class. The ProteinMPNN module in codon/utils/pmpnn.py depends only on torch and numpy and exposes the standard autoregressive sampling so that it can be driven directly; the Lightning wrapper, openfold, and wandb are not required.

Two parameters had to be recovered from the checkpoint and the training table rather than from the documentation:

- Vocabulary**.** 65 entries, comprising the 64 codons and one unknown, ordered by codon_order in codon/utils/codon_const.py. This was confirmed from the shape of the model.W_out.weight.
- Organism conditioning. An index into a 20,000-way grouping constructed for training, and not an NCBI taxon identifier. *E. coli* (NCBI 562) is index 1893, read from the 20000_grouping column of afdb_with_groupings.csv, which the authors publish alongside the weights.

Sampling used a temperature of 0.1 with a seeded decoding order. The output was validated for length, absence of internal stop codons, and correct translation.

CodonTransformer

CodonTransformer is released on PyPI and Hugging Face (adibvafa/CodonTransformer). Its pins conflict with the analysis environment, so it is run in a separate image built from baselines/codontransformer.Dockerfile with numpy<2 and transformers<4.50. The upper

bound on transformers is required because BigBird generates a path that CodonTransformer depends on, which changed in version 4.50. Invocation used organism="Escherichia coli general", deterministic=True and match_protein=True.

SolubleMPNN and MoMPNN

SolubleMPNN uses the soluble_model_weights/v_48_020.pt checkpoint shipped with ProteinMPNN, selected with --use_soluble_model. The MoMPNN checkpoints are available at github.com/Qivon7/MoMPNN; they follow the ProteinMPNN checkpoint format and load in the stock inference pipeline without modification. The solubility-aligned variant mompnn_protsol_ig_6_4_b01.ckpt was used. Both methods produce only amino acid sequences. They therefore enter the comparison as residue proposals, whose codons are then solved exactly by the codon layer.

LinearDesign

LinearDesign was built from source at github.com/LinearDesignSoftware/LinearDesign. The binary resolves its shared library using a path relative to the working directory and requires coding_wheel.txt to be in that directory.

S4. Correctness tests

Of the test suite, 72 tests pass, and 2 are skipped when an optional dependency is absent. Three tests carry the correctness argument.

Marginal argmax recovery. With an unbounded shell and only the fold-compatibility term active, the parser must return the ProteinMPNN marginal argmax sequence. This test isolates the amino-acid layer and the node weighting from the codon layer.

Exhaustive enumeration at zero budget. At $\delta = 0$, the lattice collapses to a fixed-protein codon lattice. On six-residue lattices, every synonymous coding sequence is enumerated, and the optimum returned by the parser is checked against the true maximum.

LinearDesign in the CAI limit. The lambda parameter of LinearDesign weights CAI against folding energy and defaults to zero, which corresponds to pure minimum free energy. No-exposed-weight removes the folding term, so the two objectives do not coincide in general. At lambda = 1000, the CAI term dominates, LinearDesign returns a CAI-maximal assignment, and the codon layer reproduces that assignment exactly on the same codon table. The test runs when LINEARDESIGN_BIN and LINEARDESIGN_TABLE are set, and is skipped

otherwise. Agreement with LinearDesign at $\delta = 0$ under its default settings is not available as a test, because the objectives cannot be made to coincide while the folding term is present. Two further tests fix the treatment of degenerate weights: a term with zero spread over the candidate pool is divided out rather than amplified, and a zero weight on the folding term changes the ranking without changing the reported energy.