

# Generalized design of sequence–ensemble–function relationships for intrinsically disordered proteins

---

In the format provided by the  
authors and unedited

# Supplementary Information: Generalized design of sequence-ensemble-function relationships for intrinsically disordered proteins

## Supplementary Section 1: Stochastic Gradient Estimators for Dynamical Systems

In this work, we employ trajectory reweighting for low-variance gradient estimation through simulations. Here we provide details on traditional stochastic gradient estimation for dynamical systems.

### A Stochastic Gradient Estimation

The problem of computing the gradient of an expectation of a function with respect to parameters defining the distribution that is integrated is well-studied in machine learning (see (1) for a complete review). Consider an objective  $\mathcal{F}$  of the form

$$\mathcal{F}(\theta) := \int p(x; \theta) f(x; \theta) dx = \mathbb{E}_{p(x; \theta)}[f(x; \theta)] \quad (1)$$

One is typically concerned with finding extrema of  $\mathcal{F}$ , making its gradient of interest:

$$\nabla_{\theta} \mathcal{F}(\theta) = \nabla_{\theta} \left[ \int p(x; \theta) f(x; \theta) dx \right] \quad (2)$$

$$= \int p(x; \theta) \nabla_{\theta} f(x; \theta) dx + \int \nabla_{\theta} p(x; \theta) f(x; \theta) dx \quad (3)$$

$$= \int p(x; \theta) \nabla_{\theta} f(x; \theta) dx + \int p(x; \theta) \nabla_{\theta} \log(p(x; \theta)) f(x; \theta) dx \quad (4)$$

$$= \mathbb{E}_{p(x; \theta)}[\nabla_{\theta} f(x; \theta)] + \mathbb{E}_{p(x; \theta)}[\nabla_{\theta} \log(p(x; \theta)) f(x; \theta)] \quad (5)$$

In traditional supervised learning, batches are uniformly subjected to backpropagation so  $\nabla \log(p(x; \theta)) = 0$  and  $\nabla_{\theta} \mathcal{F}(\theta) = \mathbb{E}_{p(x; \theta)}[\nabla_{\theta} f(x; \theta)]$ .

Conversely, in reinforcement learning, the distribution  $p(x; \theta)$  (defined by the policy) explicitly depends on  $\theta$  but the reward for a given state  $f(x; \theta)$  typically does not. One solution to this is to *reparameterize* the source of the stochasticity. Under the *pathwise gradient estimator*, rather than generating samples from the distribution  $p(x; \theta)$ , samples are first generated from a distribution  $p(\varepsilon)$  that is independent of  $\theta$  and only then are these generated samples transformed via a deterministic path  $g(\varepsilon, \theta)$ . Like the case of supervised learning, the reward now explicitly depends on  $\theta$  (via the sampling process) and our distribution does not depend on  $\theta$  (i.e.  $\nabla_{\theta} p(\varepsilon) = 0$ ). So, the gradient calculation acts directly on the sequence of operations that are applied to sources of randomness to yield the objective function:

$$\nabla_{\theta} \mathcal{F}(\theta) = \mathbb{E}_{p(\varepsilon)}[\nabla_{\theta} f(g(\varepsilon, \theta))] \quad (6)$$

In reinforcement learning, this is referred to as the *reparameterization trick*.

There is a second, alternative gradient estimator in reinforcement learning that does not differentiate through the sampling procedure itself but only requires that the probability of sampling each state is differentiable. Traditionally, the probability of each state is represented as the joint probability of each individual step (as determined by the policy) in the trajectory that yielded a given state. Under this *score-function gradient estimator*, one deals instead with the second term in Supplementary Equation 5:

$$\nabla_{\theta} \mathcal{F}(\theta) = \mathbb{E}_{p(x; \theta)}[\nabla_{\theta} \log(p(x; \theta)) f(x; \theta)] \quad (7)$$

This estimator is referred to as the REINFORCE algorithm in the reinforcement learning community. Intuitively, it provides signal to increase the probability of high reward states and decrease the probability of low reward states.

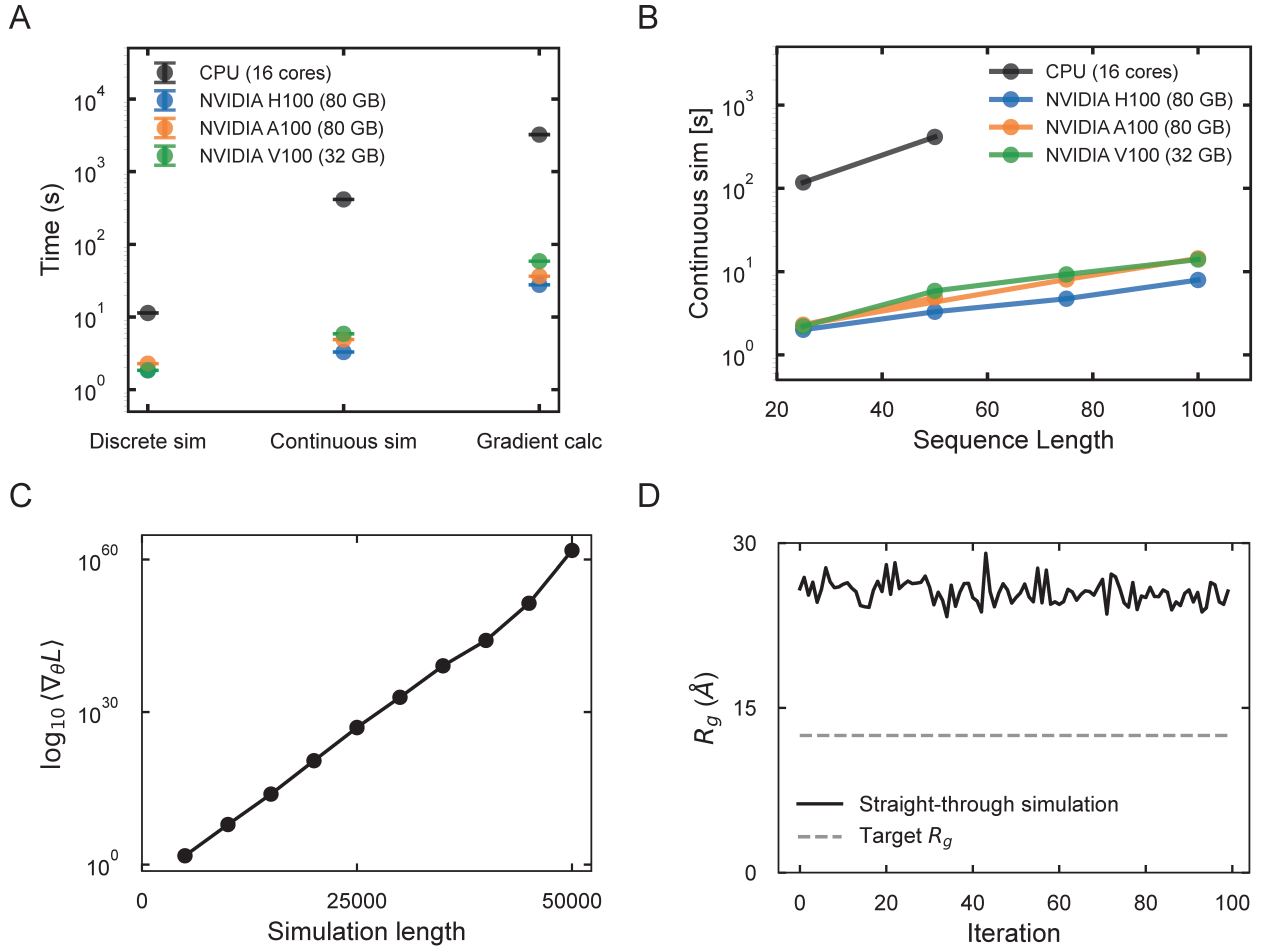
### B Differentiable Molecular Dynamics

In traditional molecular dynamics simulations, a system of  $n$  interacting bodies, typically represented by a vector  $\vec{x} \in \mathbb{R}^{6n}$  representing their positions and momenta, is iteratively propagated through time via a step function  $\mathcal{S}$ :

$$\vec{x}_{t+1} = \mathcal{S}(\vec{x}_t, \theta)$$

where  $\mathcal{S}$  depends on the energy function and numerical integration scheme and  $\theta$  are control variables. For some fixed time  $N$ , we can represent the final state  $\vec{x}_N$  as a single function

$$\mathcal{T}(\vec{x}_0) = \mathcal{S}(\cdots \mathcal{S}(\mathcal{S}(\vec{x}_0)) \cdots) = \vec{x}_N \quad (8)$$



**Supp. Fig. 1. Benchmarking a differentiable implementation of the Mpipi force field**

**A.** A comparison of simulation times for a sequence of length  $n = 50$  between forward simulations with discrete and continuous sequence representations, and a gradient calculation directly through the unrolled simulation with the continuous sequence. Each simulation consists of 50,000 timesteps of a Langevin integrator. No neighbor lists are used, i.e. nonbonded interactions are computed between all particle pairs.

**B.** A comparison of the typical forward simulations seen in A. for continuous sequences of varying lengths.

**C.** Scaling of the mean absolute value of the gradient (on a log scale) as a function of simulation length.

**D.** An optimization trajectory using gradients computed directly through the unrolled simulation of predicted  $R_g$  versus epochs.

where  $\mathcal{S}$  is applied  $N$  times and  $\vec{x}_0$  represents the initial state. Thus, a MD trajectory can be considered the result of a single numerical calculation.

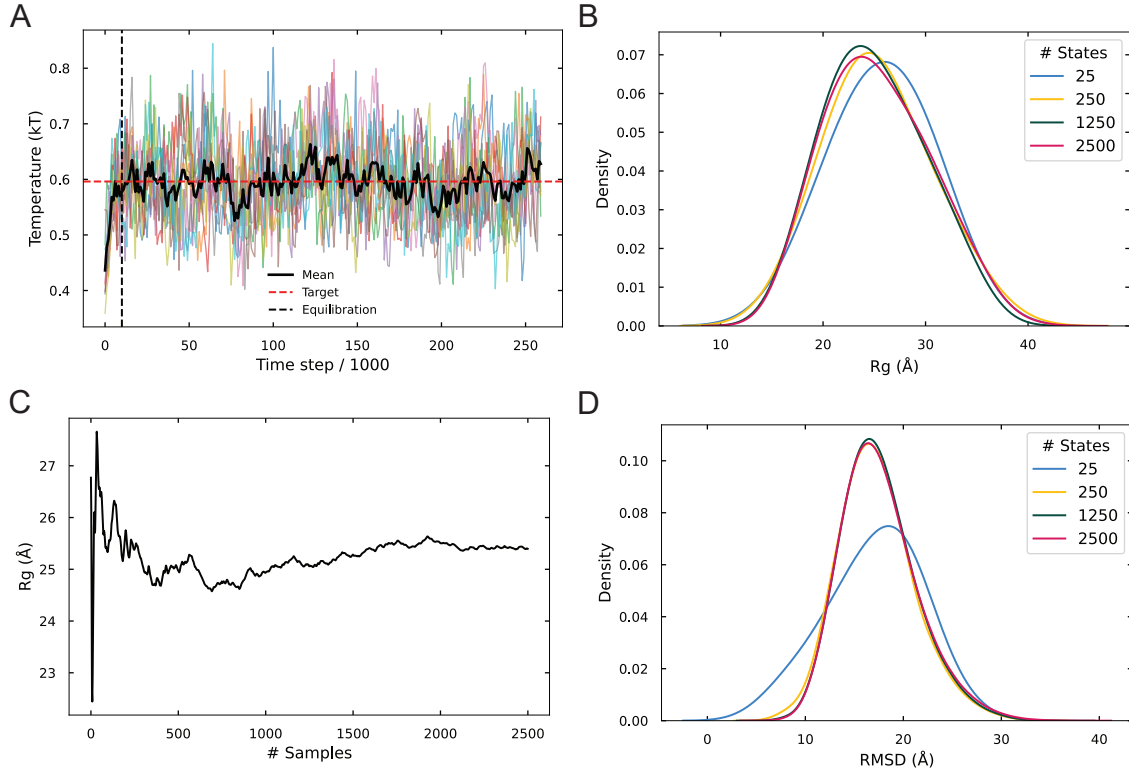
When written in an automatic differentiation framework, gradients can be computed efficiently *with respect to* this calculation. Consider some objective function defined with respect to the final state and some control parameters  $\theta$ ,  $O(\vec{x}_N, \theta)$ . Since MD trajectories are stochastic, one is typically interested in the expectation of this objective function,

$$\langle O(\vec{x}_N, \theta) \rangle_{\rho \in \mathcal{R}} = \langle O(\mathcal{T}(\vec{x}_0), \theta) \rangle_{\rho \in \mathcal{R}} \quad (9)$$

$$= \frac{1}{|\mathcal{R}|} \sum_{\rho \in \mathcal{R}} O(\mathcal{T}_\rho(\vec{x}_0), \theta) \quad (10)$$

where  $\mathcal{R}$  is a set of random seeds for initializing trajectories and  $\vec{x}_{N,\rho}$  is the final state resulting from seed  $\rho \in \mathcal{R}$ . Therefore, in the language of stochastic gradient estimators, gradients are typically computed via the reparameterization trick (c.f. (2)):

$$\nabla_\theta \mathbb{E}[O(\vec{x}_N, \theta)]_{\rho \in \mathcal{R}} \approx \langle \nabla_\theta O(\mathcal{T}_\rho(\vec{x}_0), \theta) \rangle_{\rho \in \mathcal{R}} \quad (11)$$



**Supp. Fig. 2. Sampling sufficiently representative reference ensembles**

**A.** The simulation temperature for 10 independent simulations involving 10,000 equilibration steps and 250,000 simulation steps, sampling reference states every 1,000 steps. These simulation parameters are used to sample reference states when designing sequences of length  $n = 50$  for a target  $R_g$  in Figure 2.

**B.** Distributions of sampled  $R_g$  values for a range of simulation lengths (full simulation includes 2500 reference states).

**C.** The running average of the  $R_g$  across the entire reference ensemble.

**D.** Distributions of all pairwise RMSDs between sampled reference states for a range of simulation lengths. For each unique pair of reference states, the RMSD is computed via the Kabsch algorithm (4), which finds the optimal rotation that minimizes RMSD between two point sets using singular value decomposition (SVD) of their covariance matrix.

Note that (i) objective functions can also be defined with respect to the entire trajectory itself rather than only the final state and (ii) in the case of equilibrium systems, one long simulation with states sampled at sufficiently long time intervals can be interpreted as a set of individual trajectories under standard assumptions of ergodicity, but we retain the current formalism for clarity.

Since gradients must be computed with respect to the simulation procedure itself, prohibitive numerical and memory limitations arise. In (3), Metz et al. derived the analytical gradient for a single term in Supplementary Equation 11:

$$\frac{dO_N}{d\theta} = \frac{\partial O_N}{\partial \theta} + \sum_{k=1}^N \frac{\partial O_N}{\partial \vec{x}_N} \left( \prod_{i=k}^N \frac{\partial \vec{x}_i}{\partial \vec{x}_{i-1}} \right) \frac{\partial \vec{x}_k}{\partial \theta} \quad (12)$$

where  $O_N$  represents the objective function evaluated at the final state. Importantly, the matrix of partial derivatives  $\frac{\partial \vec{x}_i}{\partial \vec{x}_{i-1}}$  is the Jacobian of the dynamical system. Only when the magnitude of all eigenvalues of this Jacobian are less than one will the resulting product be well-behaved; otherwise, the product will diverge. Moreover, as MD trajectories often require very large numbers of steps, the memory required to compute Supplementary Equation 12 can often far exceed the limitations of state of the art GPUs.

## Supplementary Section 2: Benchmarking

Our method requires that we simulate a probabilistic sequence  $\pi$  rather than a discrete sequence. For a pairwise energy function, this requires an energy calculation that is  $\mathcal{O}(20^2 n^2)$  rather than  $\mathcal{O}(n^2)$ . In Supplementary Figure 1A, we show

that this incurs only a modest computational cost for a system of size  $n = 50$  on state-of-the-art GPUs. We find an increased computational cost on CPU, likely due to decreased parallelism and therefore dominance of the  $20^2$  constant factor. Given current limitations of JAX-MD, we do not use neighbor lists in this work but the efficient usage of neighbor lists should permit the scaling of this relationship to arbitrarily longer sequences. Even without such optimizations, we find that typical forward calculations only show a modest, near-linear dependence of simulation time with increasing sequence length as seen in Supplementary Figure 1B.

Traditional differentiable MD requires the calculation of gradients through the unrolled trajectory. In Supplementary Figure 1A, we also shows how such a gradient calculation imposes a substantial time overhead to the otherwise forward simulation. Moreover, this gradient calculation is unstable as gradients explode with increasing numbers of timesteps (Supplementary Figure 1C). These gradients cannot be used for optimization even in the context of a simplified version of the most basic design problem considered in this work, i.e. designing a sequence with a target  $R_g$  *without* annealing the entropy of the probabilistic sequence (Supplementary Figure 1D).

### Supplementary Section 3: $R_g$ Optimizations

In Figures 2 and 4, we report sequence optimizations for target  $R_g$  values. This requires a definition of  $R_g$  for a probabilistic sequence. In the context of IDPs, the  $R_g$  is defined as the root mean square distance of particles from the center of mass,

$$R_g^2 = \frac{1}{n} \sum_{i=1}^n r_i^2 \quad (13)$$

where  $r_i^2$  denotes the squared distance of the  $i^{th}$  particle to the center of mass. While the particle positions are independent of the particle identities, the center of mass is not as in general the mass of a particle depends on its residue type. Consider a force field that assigns masses  $\vec{m}$  to each residue where  $\vec{m}_j$  is the mass of the  $j^{th}$  residue and  $|\vec{m}| = 20$ . Given a probabilistic sequence  $\pi$ , we define the mass of the  $i^{th}$  residue as  $\pi_i \cdot \vec{m}_i$  where  $\cdot$  denotes the dot product. For a sequence of length  $n$ , we can then define the center of mass as

$$\vec{x}_{\text{COM}} = \frac{\sum_{i=1}^n (\pi_i \cdot \vec{m}_i) \vec{x}_i}{\sum_{i=1}^n \pi_i \cdot \vec{m}_i} \quad (14)$$

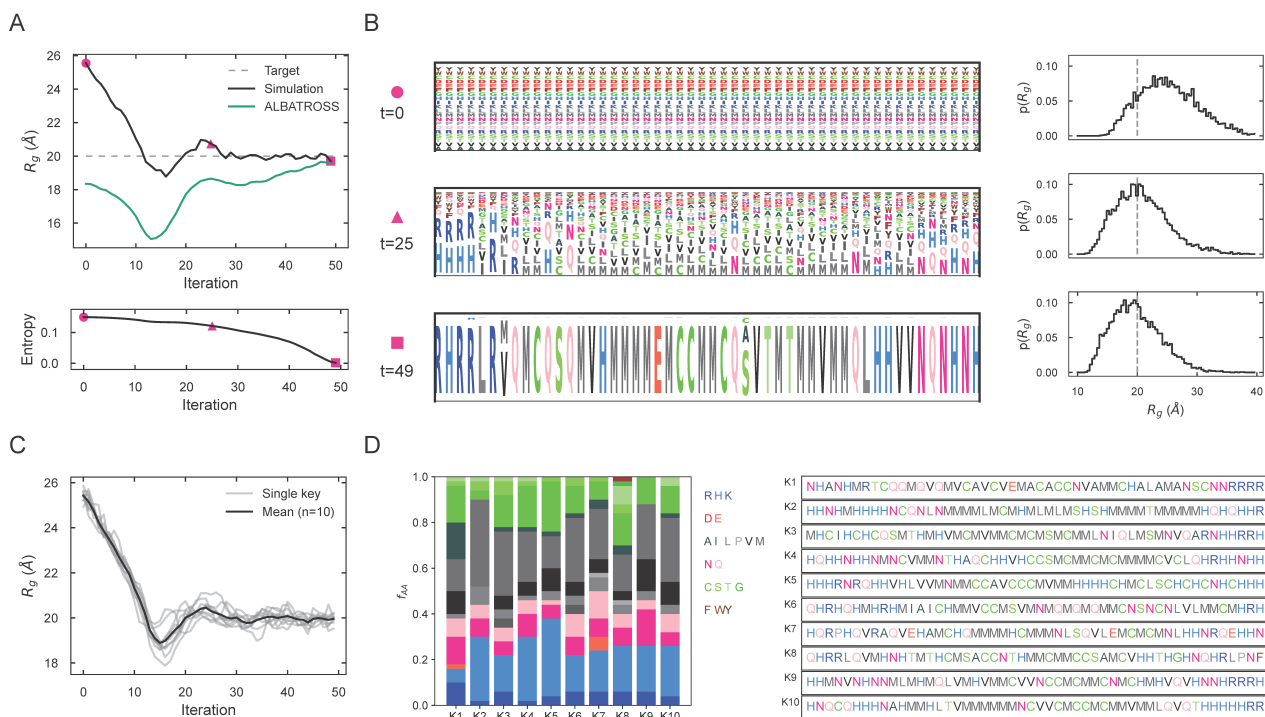
where  $\vec{x}_i$  denotes the position of the  $i^{th}$  particle.

Since  $R_g$  is defined for a single state, the  $R_g$  values reported in Figures 2 and 4 are expected values over a trajectory, i.e.  $R_g^{\text{sim}} = \mathbb{E}[R_g(\vec{x})]_{\vec{x} \sim p(\cdot; \pi)}$ . For optimization, we define the loss as  $\mathcal{L}(\pi) = \text{RMSE}(R_g^{\text{sim}}, R_g^{\text{target}})$  where  $R_g^{\text{target}}$  is the target value.

For the optimization in Figure 2B, reference ensembles were generated from 10 independent simulations consisting of 100 ps of equilibration and 2.5 ns of simulation, with snapshots sampled every 10 ps. We performed 50 iterations of gradient descent with a learning rate of 0.1. For the optimizations in Figure 2D-E, we used (i) 15 independent simulations rather than 10, (ii) simulation lengths of 5 ns rather than 2.5 ns, and (iii) 100 iterations of gradient descent rather than 50. Individual optimized sequences were validated via a more expensive calculation, consisting of 15 independent simulations each with 5 ns of equilibration and 50 ns of simulation, with a sample frequency of 100 ps.

In Figure 2, we also compare the simulated  $R_g$  to the average  $R_g$  of discrete sequences as predicted by ALBATROSS. ALBATROSS is a model developed by Lotthammer et al. that is trained on Mpipi simulations to predict single-molecule properties such as  $R_g$  and  $R_{ee}$  (5). Given a probabilistic sequence  $\pi \in \mathbb{R}^{n \times 20}$  (i.e. the parameter of a product of categorical distributions), discrete sequences can be sampled according to Equation 3 by independently sampling residue identities at each position. To compute  $\sum_{\vec{s} \in S} p(\vec{s} | \pi) O_{\vec{s}}$  (with  $O_{\vec{s}} = \mathbb{E}[O(\vec{x})]_{\vec{x} \sim p(\cdot; \vec{s})}$ ) via ALBATROSS, we sample 1000 discrete sequences and compute the average  $R_g$  as predicted by ALBATROSS. We find that 1000 samples yields a converged average. We access ALBATROSS via the `sparrow` package made available at the following link: <https://github.com/idptools/sparrow>.

Since our optimization framework relies on sampling a sufficiently representative ensemble, we also sought to evaluate the quality of our sampled ensemble. Using the simulation parameters for the  $n = 50$  optimizations, we evaluated four metrics: temperature convergence, distribution of sampled  $R_g$  values, running average of  $R_g$ , and the pairwise RMSDs between all sampled states (Supplementary Figure 2). First, we find that post-equilibration, the temperature fluctuates around the target simulation value in accordance with NVT sampling (Supplementary Figure 2A). Second, we observe a broad, converged distribution of sampled  $R_g$  values as well as a relatively stable running average (Supplementary Figure 2B-C). It is important to note that the computed average must not be perfectly converged for stable optimization as optimization operates at the level of individual microstates, increasing and decreasing the probability of individual configurations,



**Supp. Fig. 3. Flexibility of inverse design framework**

**A.** An explicit comparison of simulation predicted  $R_g$  (in Figure 2A) to ALBATROSS, a neural network trained on the Mpipi-GG force field to predict  $R_g$  from sequence, over time. The black curve represents the  $R_g$  from the simulated probabilistic sequence and the green curve represents the average  $R_g$  of discrete sequences sampled from the distribution of sequences defined by the probabilistic sequence as predicted by ALBATROSS. Highlighted points (in pink) represent the start, mid-point, and end of the optimization trajectory.

**B.** The evolution of the probabilistic sequence throughout the optimization depicted in (A) for each highlighted trajectory point shown through a WebLogo-style plot where height of the amino acid letter is proportional to its relative probability. On the right panel, the probability distribution of  $R_g$  values across the conformational ensemble is shown.

**C.** The  $R_g$  of the simulated probabilistic sequence versus epochs for ten trajectories initialized with random keys (gray lines) but with the same target  $R_g = 20$  Å and the black line represents the average profile.

**D.** The left panel represents the composition of each of the 10 optimized sequences that were obtained from different random keys (x-axis) and the height of the bar is proportional to the frequency at which a particular residue appears in the sequence. To the right, the final sequences are listed.

| n  | Target $R_g$ | Simulated $R_g$ | ALBATROSS $R_g$ |
|----|--------------|-----------------|-----------------|
| 50 | 32.5         | 32.4            | 28.5            |
| 75 | 42.5         | 41.3            | 39.0            |
| 75 | 45           | 43.4            | 37.8            |
| 75 | 47.5         | 46.1            | 39.7            |
| 75 | 50           | 48.6            | 39.0            |

**Supp. Table 1.**  $R_g$  optimizations for which ALBATROSS underpredicts the simulated  $R_g$ .

rather than the computed average. We validate all optimizations with substantially longer simulations in which case we observe high-resolution convergence in the average  $R_g$ . Third, upon computing all pairwise RMSDs between sampled states, we observe a converged distribution indicating broad diversity between samples, with an average pairwise RMSD of 17.6 Å (Supplementary Figure 2D). Beyond these summary statistics under the simulation parameters for optimization, the convergence of our optimizations and the subsequent validation with relatively expensive forward simulations validates the quality of the gradient signal enabled by these ensembles.

## Supplementary Section 4: $R_{ee}$ Optimizations

In Figure 2, we also report sequences optimized for target values of  $R_{ee}$ . Unlike  $R_g$ ,  $R_{ee}$  is only a function of particle positions and therefore has the same definition for a continuous sequence as for a discrete sequence. Specifically, for a sequence of length  $n$ ,  $R_{ee}$  is defined as simply the distance between the first and last particles:

$$R_{ee} = d(\vec{x}_1, \vec{x}_n) \quad (15)$$

where  $d$  denotes the Euclidean distance. The loss is defined similarly as in the case of  $R_g$ , i.e.  $\mathcal{L}(\pi) = \text{RMSE}(R_{ee}^{\text{sim}}, R_{ee}^{\text{target}})$  where  $R_{ee}^{\text{target}}$  is the target value and  $R_{ee}^{\text{sim}} = \mathbb{E}[R_{ee}(\vec{x})]_{\vec{x} \sim p(\cdot; \pi)}$ .

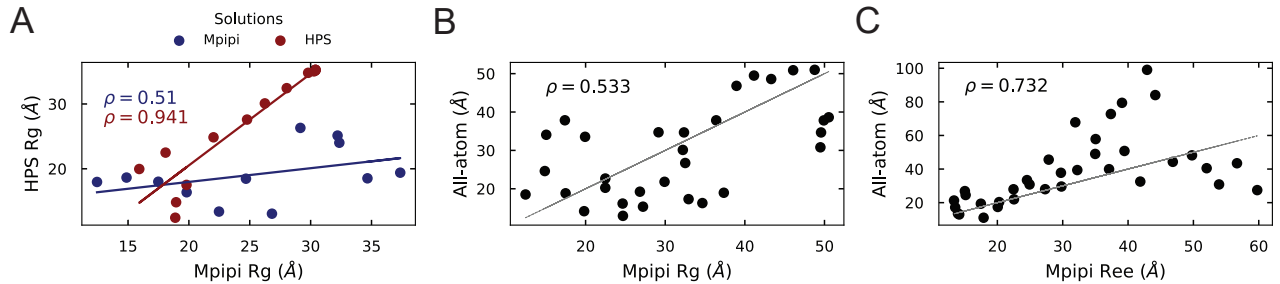
Both the collection of reference ensembles throughout optimization and the validation of designed sequences were performed using the same simulation parameters as the  $R_g$  optimizations in Figure 2D-E.

## Supplementary Section 5: Cross-Validation with Alternative Force Fields

When designing sequences with target values of  $R_g$  and  $R_{ee}$ , we used two different underlying force fields, i.e. Mpipi and HPS. One natural question is the extent to which solutions under one force field generalize to alternative force fields. Note that since each force-field typically incorporates different biophysical/statistical priors, exact agreement is not expected.

First, as a simple test, we correlated the simulated  $R_g$ s under both HPS and Mpipi for all sequences of length  $n = 50$  designed for target  $R_g$  values under both force fields. We find broad agreement between the two force fields, with a stronger correlation arising from the solutions designed under HPS (Supplementary Figure 4A). One candidate explanation for the decreased correlation when evaluating sequences designed under Mpipi is that Mpipi models more complex physical interactions (for instance,  $\pi$ - $\pi$  interactions), and therefore solutions that rely on such physics may not correlate with HPS.

Next, we sought to evaluate the extent to which sequences designed with respect to a coarse-grained model agree with fine-grained, all-atom simulations. We use the GB99dms all-atom force field developed by Greener, which combines the a99SB-disp protein force field with the GBNeck2 implicit solvent model and was fit using differentiable molecular simulation to better capture the behavior of IDPs (6). Estimates of  $R_g$  and  $R_{ee}$  were obtained via 3 independent 100 ns simulations with 0.5 ns equilibration time and configurations sampled every 0.5 ns. We used a Langevin integrator (i.e. LangevinMiddle in OpenMM version  $\geq 8.0.0$  with a step size of 2 fs. Initial conformations were obtained from the ESM Metagenomic Atlas (7). We correlate the simulated  $R_g$ s and  $R_{ee}$ s of sequences designed with respect to Mpipi with the simulated values using GB99dms. We find broad agreement between the two force fields, with increased correlation between  $R_{ee}$  compared to  $R_g$  values (Supplementary Figure 4B-C).



**Supp. Fig. 4. Evaluating solutions designed with Mpipi and HPS with alternative force fields.**

**A.** Correlations between simulated Mpipi and HPS  $R_g$  values across all sequences designed with respect to both force fields.

**B.** Correlation between Mpipi and GB99dms all-atom  $R_g$  values across  $n = 50$  and  $n = 75$  sequences designed with respect to Mpipi.

**C.** Correlation between Mpipi and GB99dms all-atom  $R_{ee}$  values across  $n = 50$  and  $n = 75$  sequences designed with respect to Mpipi.

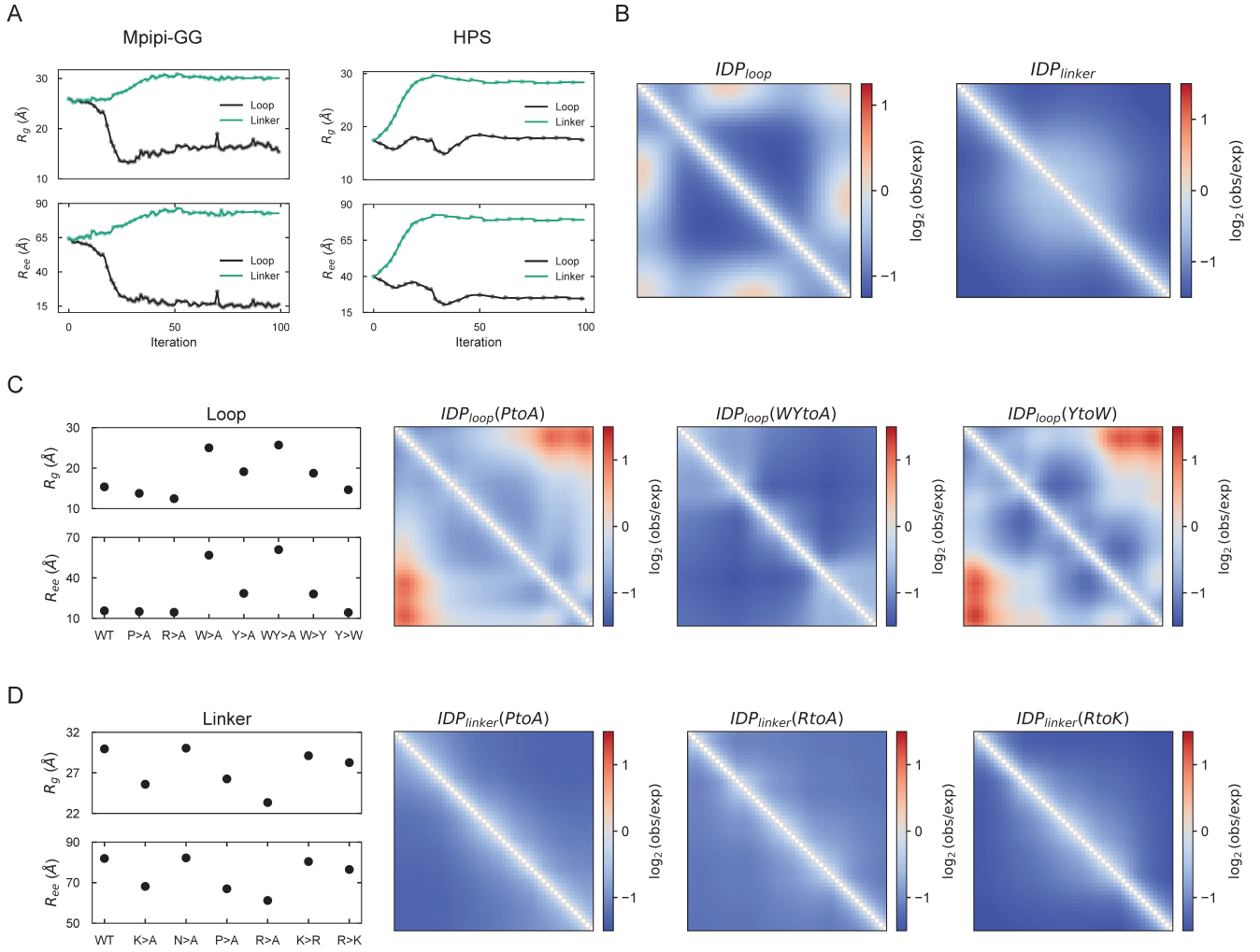
## Supplementary Section 6: Loop and Linker Optimizations

In Figures 3C.I-II, we present the optimizations of loops and linkers – IDPs for which  $R_g \gg \frac{R_{ee}}{\sqrt{6}}$  and  $\frac{R_{ee}}{\sqrt{6}} \gg R_g$ , respectively. For a given probabilistic sequence  $\pi$ , we define the loss functions as

$$\mathcal{L}_{\text{loop}}(\pi) = \frac{R_{ee}^{\text{sim}}}{\sqrt{6}} - R_g^{\text{sim}} \quad (16)$$

and

$$\mathcal{L}_{\text{linker}}(\pi) = R_g^{\text{sim}} - \frac{R_{ee}^{\text{sim}}}{\sqrt{6}} \quad (17)$$



**Supp. Fig. 5. Probing mechanisms of loop and linker assembly**

**A.** Each panel depicts the convergence of  $R_g$  and  $R_{ee}$  over the optimization iterations for representative loop (black lines) and linker (green lines) optimizations with  $n = 50$  for the Mpipi-GG (left panels) and HPS (right panels) force-fields. **B.** The panels depict normalized contact maps over representative trajectories for the loop (left) and linker (right) solutions under the HPS force field, analogous to the Mpipi-GG force field depicted in Figure 3. For contact frequencies, red/blue regions represent higher/lower expected frequencies when contrasted with an ideal polymer of identical length.

**C.** The  $R_g$  (top) and  $R_{ee}$  (bottom) are shown for both the optimized loop solution (WT or wild-type) and a set of mutational scans. On the right, the corresponding contact maps (as in B.) are shown for a subset of mutants.

**D.** The  $R_g$  (top) and  $R_{ee}$  (bottom) are shown for both the optimized linker solution (WT or wild-type) and a set of mutational scans. On the right, the corresponding contact maps (as in B.) are shown for a subset of mutants.

Supplementary Figure 5 depicts the convergence of the  $R_g$  and  $R_{ee}$  values for the representative optimizations depicted in Figures 3C.I-II. Supplementary Table 2 lists the results of the mutational analyses summarized in the insets of Figures 3C.I-II.

Simulations were performed using the same parameters as for the  $R_g$  and  $R_{ee}$  optimizations in Figure 2D-E.

## Supplementary Section 7: Sequence Constraints

### A Constraint Activation Function

Consider a function  $C: \mathbb{R}^{n \times 20} \rightarrow \mathbb{R}$  that accepts a probabilistic sequence  $\pi$  as input and returns a scalar value. We wish to design a sequence that satisfies a minimum value  $C_{\min}$  of this function. Following the work of Krueger and Ward (8),

| Sequence             | R <sub>g</sub> | R <sub>ee</sub> | Loss |
|----------------------|----------------|-----------------|------|
| Solution             | 15.4           | 15.6            | -9.0 |
| P>A                  | 13.8           | 15.0            | -7.6 |
| R>A                  | 12.5           | 14.5            | -6.5 |
| W>A                  | 25.0           | 56.8            | -1.8 |
| Y>A                  | 19.1           | 28.5            | -7.5 |
| W>Y                  | 18.7           | 28.1            | -7.3 |
| Y>W                  | 14.6           | 14.3            | -8.8 |
| Min. R <sub>ee</sub> | 11.7           | 13.8            | -6.1 |
| Max. R <sub>g</sub>  | 36.2           | 93.9            | 2.1  |

(a) Loop

| Sequence             | R <sub>g</sub> | R <sub>ee</sub> | Loss |
|----------------------|----------------|-----------------|------|
| Solution             | 30.0           | 81.9            | -3.5 |
| K>A                  | 25.6           | 68.1            | -2.2 |
| N>A                  | 30.0           | 82.2            | -3.5 |
| P>A                  | 26.2           | 66.9            | -1.1 |
| R>A                  | 23.4           | 61.1            | -1.6 |
| K>R                  | 29.1           | 80.4            | -3.7 |
| R>K                  | 28.3           | 76.5            | -3.0 |
| Max. R <sub>ee</sub> | 37.2           | 96.6            | -2.2 |
| Min. R <sub>g</sub>  | 10.6           | 17.9            | 3.3  |

(b) Linker

**Supp. Table 2.** Mutational analysis for the loop and linker optimizations. Loss column for loop and linker tables represents  $\frac{R_{ee}}{\sqrt{6}} - R_g$  and  $R_g - \frac{R_{ee}}{\sqrt{6}}$ , respectively.

we define a ReLU function that increases sharply below the minimum value and increases slowly above this threshold:

$$\phi_C(C_\pi) = \begin{cases} m_1 C_\pi + (1 - m_1 C_{\min}), & \text{for } C_\pi < C_{\min} \\ m_2 C_\pi + (1 - m_2 C_{\min}), & \text{for } C_\pi \geq C_{\min} \end{cases} \quad (18)$$

where  $C_\pi = C(\pi)$  and  $m_1$  and  $m_2$  are hyperparameters with  $m_1 < m_2 \leq 0$ . Note that  $\phi_C$  is defined such that  $\phi_C(C_{\min}) = 1.0$  and values  $C_\pi < C_{\min}$  are strongly penalized while values  $C_\pi \geq C_{\min}$  are mildly rewarded.

For a set of such constraint functions  $C_i$  and their corresponding activation functions  $\phi_{C_i}$ , we define our objective function as the geometric mean of the baseline loss function and the activated constraints,

$$\mathcal{L}^{\text{const}}(\pi) = \mathcal{L}(\pi) \times \prod_i \phi_{C_i}(C_i(\pi)) \quad (19)$$

where  $\mathcal{L}(\pi)$  is the RMSE between the simulated and target observables in the case of  $R_g$  and  $R_{ee}$ . There are alternative methods for applying such constraints – for example, projected gradient methods – though such methods would be complicated by our annealing of the sequence entropy and we find that the geometric mean works well in practice.

## B Disorder Constraint

Since sequence entropy does not provide a reliable measure of disorder for probabilistic sequences, we turn to Metapredict, a machine learning model trained to predict consensus sequence disorder (9). Consider Metapredict as a function  $\text{MP} : \mathbb{R}^{n \times 20} \rightarrow \mathbb{R}^n$  that maps a probabilistic sequence to an expected disorder at each position. We therefore define the disorder of a probabilistic sequence as the average predicted disorder, i.e.

$$D(\pi) = \frac{1}{n} \sum_{i=1}^n \text{MP}(\pi)_i \quad (20)$$

Since we use Metapredict v2, which predicts normalized measures of disorder,  $D(\pi) \in [0, 1]$ . Note that Metapredict was only trained on discrete sequences and therefore was not intended for use with probabilistic sequences.

Though Metapredict provides a more reliable proxy for sequence disorder than sequence entropy, it still underpredicts the expected disorder for high entropy sequences and in practice we only wish to demand that the final sequence is disordered. Thus, we anneal  $D_{\min}$  from 0.2 to 0.8 throughout the first 80% of the optimization iterations. For  $\phi_D$ , we define  $m_1 = -1000$  and  $m_2 = -0.01$ .

Reference states for the optimization in Figure 4B were collected using the same parameters as in Figure 2B. Reference states for the optimizations in Figure 4C-E were collected using the same parameters as in Figure 2D-E. Designed sequences in Figure 4C and Figure 4 were obtained using 150 and 200 iterations of gradient descent, respectively. Forward simulations of designed sequences to compute the statistics presented in Figure 4C-E were performed using the same parameters as in the validation of designed sequences in Figure 2.

## C Charge Distribution Constraints

In Figure 4, we design IDP sequences with a target distribution of positively and negatively charged residues. Therefore, we require a definition for the ratio of a probabilistic sequence that is a given residue type. For clarity, we restrict attention

to the case of positively charged residues.

For a discrete sequence  $\vec{s}$  of length  $n$ , the ratio of positively charged residues is defined as

$$R_+(\vec{s}) = \frac{1}{n} \sum_{i=1}^n \delta_+(\vec{s}_i) \quad (21)$$

where  $\delta_+(\vec{s}_i) = 1$  if  $\vec{s}_i$  is a positively charged residue and  $\delta_+(\vec{s}_i) = 0$  otherwise. Since the distribution of residues at each position is normalized in a probabilistic sequence, this definition can be generalized to such sequences:

$$\mathbb{E}[R_+(\vec{s})]_{\vec{s} \sim \pi} = \sum_{\vec{s} \in S} p(\vec{s}|\pi) R_+(\vec{s}) \quad (22)$$

$$= \sum_{\vec{s} \in S} p(\vec{s}|\pi) \left( \frac{1}{n} \sum_{i=1}^n \delta_+(\vec{s}_i) \right) \quad (23)$$

$$= \frac{1}{n} \sum_{i=1}^n \sum_{\vec{s} \in S} p(\vec{s}|\pi) \delta_+(\vec{s}_i) \quad (24)$$

$$= \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^{20} \pi_{ij} \delta_+(\vec{s}_i) \quad (25)$$

$$= \frac{1}{n} \sum_{i=1}^n \pi_i \cdot \vec{\delta}_+ \quad (26)$$

where  $\vec{\delta}_+ \in \mathbb{R}^{20}$  is a one-hot vector denoting whether or not a given amino acid is positively charged and  $\cdot$  denotes the dot product.

As for disorder constraints, we apply an activation function  $\phi_R$  to the computed ratios. For the ratios of both positively and negatively charged residues, we set  $R_{min} = 0.495$  to relax the search space, and define  $m_1 = -1000$  and  $m_2 = -0.01$ . In practice all presented sequences satisfy the constraints within 5% error. We initialize logits corresponding to a value of  $\pi$  such that each position has a 0.495 probability of being both positively and negatively charged, and uncharged residue types uniformly distributed over the remaining cumulative probability  $1.0 - 2 \times 0.495 = 0.01$ .

| Sensor Type     | Response Type | $R_g^{lo}$ | $R_g^{hi}$ | $\mathcal{L}_{oss}$ |
|-----------------|---------------|------------|------------|---------------------|
| Salt            | Contractor    | 23.1       | 14.1       | -9.0                |
| Salt            | Expander      | 14.4       | 26.6       | -12.2               |
| Phosphorylation | Contractor    | 24.4       | 23.3       | -1.1                |
| Phosphorylation | Expander      | 16.7       | 19.4       | -2.7                |
| Temperature     | Contractor    | 32.0       | 31.3       | 0.7                 |
| Temperature     | Expander      | 14.2       | 19.7       | -5.5                |

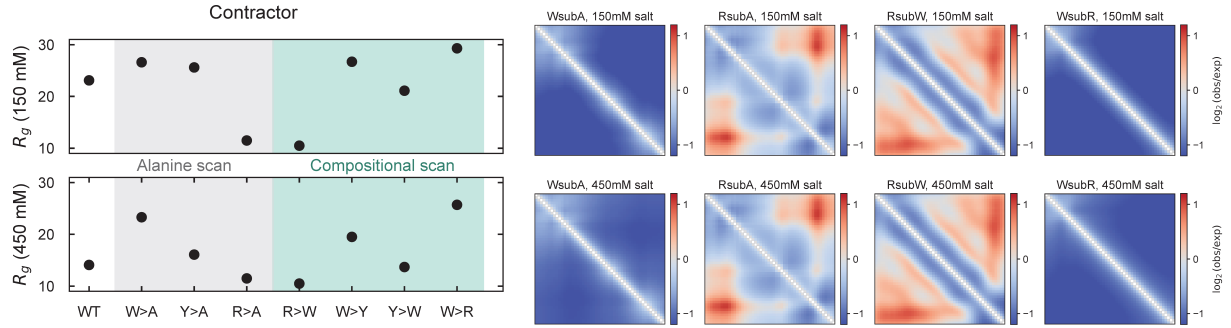
**Supp. Table 3.** Optimized sensors of length  $n = 50$  for a range of sensor and response types. For a salt sensor,  $R_g^{lo}$  and  $R_g^{hi}$  correspond to 150 mM and 450 mM, respectively. For a phosphorylation sensor,  $R_g^{lo}$  and  $R_g^{hi}$  correspond to the 25<sup>th</sup> position fixed as serine (S) and Glutamic acid (E), respectively. For a temperature sensor,  $R_g^{lo}$  and  $R_g^{hi}$  correspond to 293.15 K (20 C) and 363.15 K (90 C), respectively.

## D Overparameterization

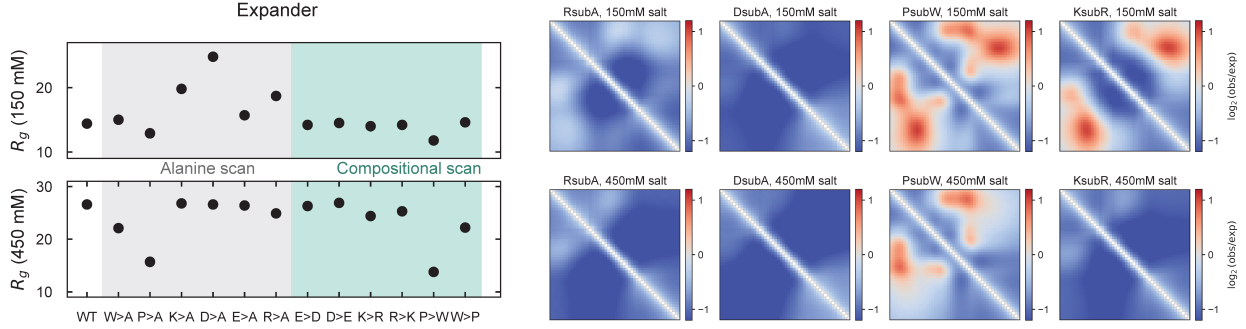
In Figure 4, we overparameterize the search space by optimizing over the weights of a neural network that outputs a  $n \times 20$  matrix of logits. We use a fully-connected architecture for all networks with 6 layers of 4000 nodes each. We apply a Leaky ReLU activation function between each layer. We pretrain the network to output a target set of logits corresponding to a target probabilistic sequence (i.e. a uniform-distributed probabilistic sequence or one with a target distribution of charged residues) Given a target initial pseq  $\pi^{init}$ , we define the following pretrain loss:

$$\mathcal{L}_{pretrain}(\theta) = \frac{1}{4n} \sum_{ij} 100 \cdot (\pi_{ij}^{pred} - \pi_{ij}^{init})^2 \quad (27)$$

A



B



**Supp. Fig. 6. Mutational analyses for the optimized salt sensors**

**A-B.** For the salt sensing contractor (A) and expander (B) reported in Figure 5, we perform alanine scanning (gray background) as well as rational compositional mutations (green background) informed by the underlying sensor mechanism. For each of the sensors, we report the effect of particular mutations on  $R_g$  at low (top subpanel) and high salt (bottom subpanel). On the right, normalized contact frequencies are shown for particular mutants for low (top subpanel) and high salt (bottom subpanel). For contact frequencies, red/blue regions represent higher/lower expected frequencies when contrasted with an ideal polymer of identical length.

where  $\pi^{pred} = \text{softmax}(\text{NN}_{\theta}(k))$ ,  $\theta$  are the weights of the neural network, and  $\text{NN}_{\theta}(k)$  denotes the output of the neural network with weights  $\theta$  and a fixed random seed as input. We use an Adam optimizer with a learning rate of  $10^{-5}$  for pretraining.

## Supplementary Section 8: Sensor Optimizations

Below, we provide simulation and optimization details for each type of sensor design. In all cases, reference states for optimization were collected using the same parameters as in Figure 2D-E and we applied 100 iterations of gradient descent. Forward simulations of the designed sequences were performed using the same parameters as in the validation of designed sequences in Figure 2.

### A Salt Sensors

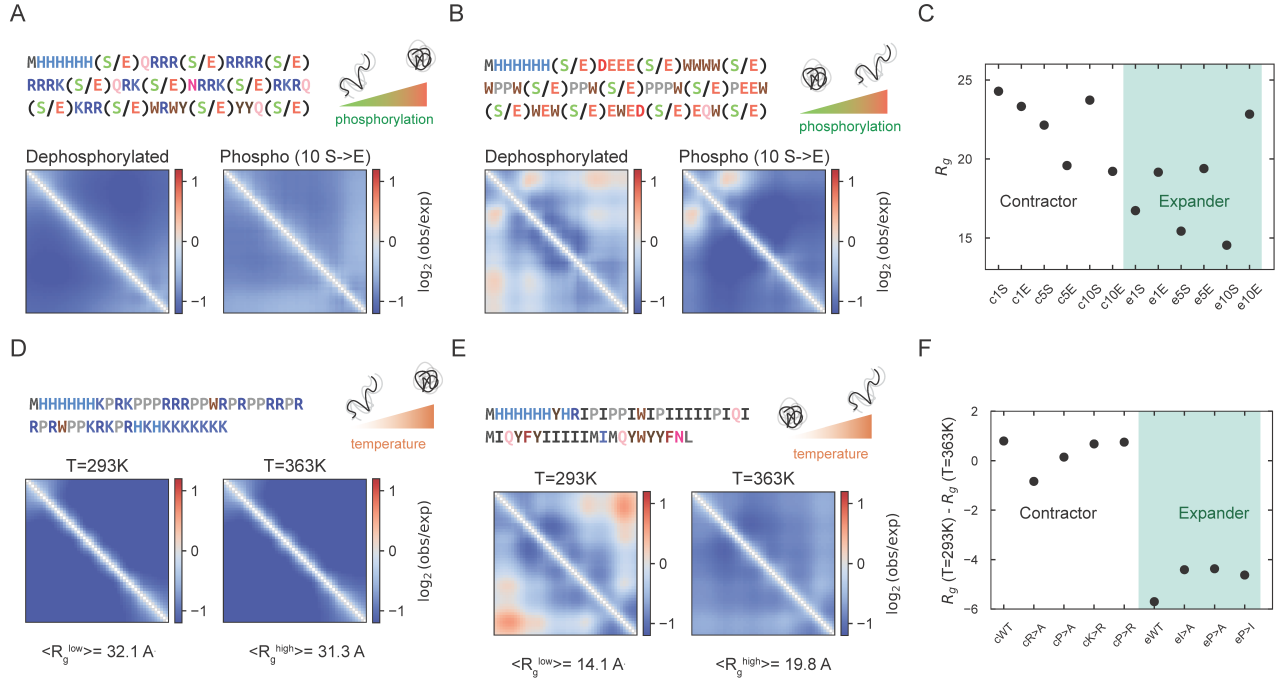
In Figure 5, we design IDPs that expand or contract upon the addition of salt. In Mpipi, a single Debye length  $\kappa$  was used to reproduce behavior of IDPs at a salt concentration of 150 mM. To model the effects of increased salt, we followed Debye-Huckel theory in which

$$\kappa^{-1} = \sqrt{\frac{\varepsilon_R \varepsilon_0 k T}{2 e^2 I}} \quad (28)$$

where  $I$  is the ionic strength,  $\varepsilon_0$  is the permittivity of free space,  $\varepsilon_R$  is the dielectric constant, and  $e$  is the elementary charge. Thus, given the default Debye length in Mpipi  $\kappa^{150}$ , we obtain the Debye length for an arbitrary salt concentration  $I$  (expressed in mM) as

$$\kappa^I = \kappa^{150} \frac{\sqrt{I/1000}}{\sqrt{150/1000}} \quad (29)$$

More generally, salt concentration can affect simulation parameters in multiple ways (for instance, the dielectric permittivity) and such effects can be easily accommodated in our framework through empirical models.



**Supp. Fig. 7. Phosphorylation and temperature sensing IDPs**

**A-B.** Optimized contractor and expander sequences with normalized contact frequency maps without (A) and with (B) phosphorylation computed from representative trajectories. For these solutions, the IDPs comprise 10 phosphosites i.e., Serines that are roughly equispaced, and highlighted in the sequence. For contact frequencies, red/blue regions represent higher/lower expected frequencies when contrasted with an ideal polymer of identical length.

**C.** The change in  $R_g$  for optimized phosphorylation sensors that exploit different numbers of phosphosites for contractors (white background) and expanders (green background). The magnitude of the effect is generically larger with more phosphosites - the axis label denotes the type of sensor (c/e), the number of serines (1/5/10), and the phosphorylation status (S/E).

**D-E.** Optimized contractor and expander sequences with normalized contact frequency maps at low (D) and high (E) temperatures. For contact frequencies, red/blue regions represent higher/lower expected frequencies than an ideal polymer of identical length.

**F.** The change in  $R_g$  for optimized temperature sensors for contractors (white background) and expanders (green background) upon mutation of key residues. The axis label denotes the type of mutant.

We formulate the sensor design problem similar to that for the design of loops and linkers, except that the loss function involves expectations from two distinct ensembles. We define two salt concentrations  $I^{lo}$  and  $I^{hi}$ , corresponding to Debye lengths  $\kappa^{lo}$  and  $\kappa^{hi}$ . For a given probabilistic sequence  $\pi$ , we compute the  $R_g$  in each ensemble, denoted  $R_g^{lo}$  and  $R_g^{hi}$ , and define the loss for the contractor as

$$\mathcal{L}_{contractor}(\pi) = R_g^{hi} - R_g^{lo} \quad (30)$$

and the loss for the expander as

$$\mathcal{L}_{expander}(\pi) = R_g^{lo} - R_g^{hi} \quad (31)$$

As a demonstration of the flexibility of our method, we impose the constraint that the sequence begins with a M-His6 motif. This represents an experimentally-relevant constraint as the start codon is required for expression and the His6 tag is commonly used for protein purification. We impose this constraint by setting the first seven rows of the probabilistic sequence (computed via  $\pi = \text{softmax}(\lambda/\tau)$  where  $\lambda$  are logits and  $\tau$  is a temperature parameter) to a  $n \times 7$  one-hot sequence representing this motif.

We optimize contractors and expanders for lengths  $n = 50$ ,  $n = 75$ , and  $n = 100$ . For  $n = 50$ , we also conduct a mutational analysis of our solutions to evaluate both the effect of individual residue types via alanine substitutions, and the Pareto front via substitution of residues with residue types similar in their electrostatic properties.

|                         |                 |      |      |      | Sequence                        | $R_g^{\text{lo}}$ | $R_g^{\text{hi}}$          | $\mathcal{L}_{\text{oss}}$ |       |
|-------------------------|-----------------|------|------|------|---------------------------------|-------------------|----------------------------|----------------------------|-------|
|                         |                 |      |      |      | Solution                        | 14.4              | 26.6                       | -12.2                      |       |
| Sequence                |                 |      |      |      | $R_g^{\text{lo}}$               | $R_g^{\text{hi}}$ | $\mathcal{L}_{\text{oss}}$ |                            |       |
| Solution                |                 |      |      |      | 23.1                            | 14.1              | -9.0                       |                            |       |
| Alanine Scanning        | W>A             | 26.6 | 23.3 | -3.3 | Alanine Scanning                | W>A               | 15.0                       | 22.1                       | -7.1  |
|                         | Y>A             | 25.6 | 16.1 | -9.5 |                                 | P>A               | 12.9                       | 15.7                       | -2.8  |
|                         | R>A             | 11.5 | 11.5 | 0.0  |                                 | K>A               | 19.8                       | 26.8                       | -7.0  |
|                         |                 |      |      | D>A  |                                 | 24.8              | 26.6                       | -1.8                       |       |
|                         |                 |      |      | E>A  |                                 | 15.7              | 26.4                       | -10.7                      |       |
| Conformational Scanning | R>W             | 10.5 | 10.5 | 0.0  | R>A                             | 18.7              | 24.9                       | -6.2                       |       |
|                         | W>Y             | 26.7 | 19.5 | -7.2 | Conformational Scanning         | E>D               | 14.2                       | 26.3                       | -12.1 |
|                         | Y>W             | 21.1 | 13.7 | -7.4 |                                 | D>E               | 14.5                       | 26.9                       | -12.4 |
|                         | W>R             | 29.3 | 25.7 | -3.6 |                                 | K>R               | 14.0                       | 24.4                       | -10.0 |
|                         | R <sub>50</sub> | 30.4 | 26.8 | -3.6 |                                 | R>K               | 14.2                       | 25.3                       | -11.1 |
| (a) Contractor          |                 |      |      | P>W  |                                 | 11.8              | 13.8                       | -2.0                       |       |
|                         |                 |      |      |      | W>P                             | 14.6              | 22.2                       | -7.6                       |       |
|                         |                 |      |      |      | E <sub>25</sub> K <sub>25</sub> | 13.4              | 15.6                       | -2.2                       |       |
|                         |                 |      |      |      | (b) Expander                    |                   |                            |                            |       |

**Supp. Table 4.** Mutational analysis via alanine substitutions and rational substitutions for the optimized salt sensors depicted in Figure 5. Loss column for contractor and expander tables represents  $R_g^{hi} - R_g^{lo}$  and  $R_g^{lo} - R_g^{hi}$ , respectively.

## B Phosphorylation Sensors

We next sought to design IDPs that contract or expand upon phosphorylation rather than in the presence of salt. Since phosphorylation is not explicitly modelled in Mpipi, we choose a residue pair representative of a phosphorylation event and modelled phosphorylation as the transition from one residue to the other; for our purposes, we considered the phosphorylation of serine (S) to Glutamic acid (E). For a given length  $n$ , we choose a phosphorylation position  $1 \leq i_{phos} \leq n$  and model the dephosphorylated and phosphorylated ensembles by explicitly setting the  $i_{phos}^{th}$  residue to be (a one-hot vector representing) S and E, respectively. The loss function is defined as in Supplementary Equations 30 and 31 where  $R_g^{lo}$  and  $R_g^{hi}$  correspond to the dephosphorylated and phosphorylated ensembles, respectively, and we similarly impose a M-His6 prefix.

## C Temperature Sensors

We also design IDPs that contract or expand in response to changes in temperature. The optimization problem is defined as above with  $R_g^{lo}$  and  $R_g^{hi}$  corresponding to 293.15 K (20 C) and 363.15 K (90 C), respectively, and we again impose a M-His6 prefix. For simplicity, we do not modify  $\kappa$  in accordance with the change in  $kT$ .

## Supplementary Section 9: Binder Optimizations

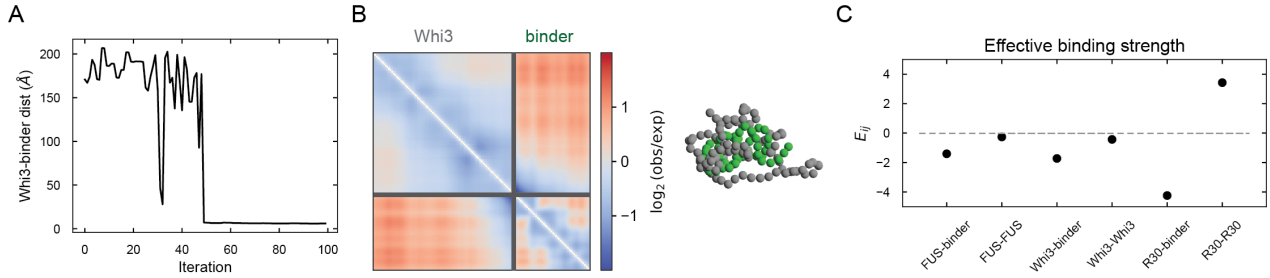
In Figure 6, we present the optimization of binder sequences for a fixed substrate. We use a simple formulation of the binder design problem in which we optimize for the binder sequence that minimizes the interstrand distance. The interstrand distance  $r_{com}$  is defined as the distance between the center of masses  $\vec{x}_{com}^{binder}$  and  $\vec{x}_{com}^{substrate}$ , each defined following Supplementary Equation 14. To maximize the probability of sampling configurations with low interstrand distance, we apply a bias potential

$$U^{bias}(r_{com}) = \begin{cases} k(r_{com} - r_{max})^2, & \text{if } r_{com} > r_{max} \\ 0, & \text{otherwise} \end{cases} \quad (32)$$

To correct this bias, we redefine the probability of a state in DiffTRE ( $p_\theta(\vec{x}_i)$  in Equation 20) following the standard umbrella sampling correction:

$$p_\theta(\vec{x}_i) = \frac{1}{\omega_i} \exp(-\beta(U_\theta(\vec{x}_i))) \quad (33)$$

where  $\omega_i = \exp(-\beta(U_\theta^{bias}(\vec{x}_i)))$ . For the optimization depicted in Figure 6, we design a binder of length  $n = 50$  and set  $k = 1$  and  $r_{max} = 300$ .



### Supp. Fig. 8. Probing IDP binder-substrate interactions

**A.** The optimization of a binder ( $n = 50$ ) for Whi3 ( $n = 93$ ) is depicted by convergence to low interstrand distance over the trajectory.

**B.** Normalized contact frequency map for the optimized Whi3 binder, highlighting both intramolecular and intermolecular interactions. Red/blue regions represent higher/lower expected frequencies when contrasted with an ideal polymer of total length binder + substrate. On the right, a representative bound snapshot of binder (green) and substrate (grey) is depicted.

**C.** Computed effective interaction coefficients ( $E_{ij}$ , units of  $nm^3$ ) between species  $i, j$  are plotted comparing optimized binder-substrate interactions with substrate-substrate interactions. More negative values represent stronger interactions.

To demonstrate the flexibility of our method, we optimize binders for two additional sequences: (1) a positively charged homopolymer (polyR) and (2) the low complexity region of Whi3 (see Supplementary Figure 8). For PolyR, we design a binder of length  $n = 30$  and set  $k = 10$  and  $r_{max} = 150$ . For Whi3-LC, we design a binder of length  $n = 50$  and set  $k = 1$  and  $r_{max} = 250$ .

We then compute effective interaction coefficients to estimate the strength of interactions between substrates and ligands, as well as homotypic substrate interactions. As reported in (10), and subsequently in a related paper (11), such coefficients broadly correlate with experimental or simulation derived interaction coefficients and multicomponent condensation. Specifically, we compute the pairwise dimer coefficient ( $B_{ij}^{rdp}$  as defined in (10)) between two biomolecular species  $i$  and  $j$ , and report normalized interactions  $E_{ij} = \frac{B_{ij}}{n_i n_j}$  in units of  $nm^3$ .

For the PolyR substrate, we sample reference states for optimization using 10 independent simulations each consisting of 250 ps of equilibration and 7.5 ns of simulation, with conformations sampled every 10 ps. For Whi3-LC and FUS-LC, we sample reference states using 8 independent simulations each consisting of 10 ns of simulation time with conformations sampled every 10 ps. We use 500 ps and 250 ps of equilibration time for Whi3-LC and FUS-LC, respectively. We validated each designed sequence using a forward simulation with periodic boundaries and the binder and substrate initialized in an unbound configuration. For each design, we evaluated a range of box sizes up to 500 Å and found that our designed binders reliably bound the target substrate. Validations for each design involved 10 independent simulations each consisting of 10 ns of equilibration and 250 ns of simulation, sampling conformations every 100 ps.

1. Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte carlo gradient estimation in machine learning. *The Journal of Machine Learning Research*, 21(1):5183–5244, 2020.
2. Megan C Engel, Jamie A Smith, and Michael P Brenner. Optimal control of nonequilibrium systems through automatic differentiation. *Physical Review X*, 13(4):041032, 2023.
3. Luke Metz, C Daniel Freeman, Samuel S Schoenholz, and Tal Kachman. Gradients are not all you need. *arXiv preprint arXiv:2111.05803*, 2021.
4. Wolfgang Kabsch. A solution for the best rotation to relate two sets of vectors. *Foundations of Crystallography*, 32(5):922–923, 1976.
5. Jeffrey M. Lotthammer, Garrett M. Ginell, Daniel Griffith, Ryan J. Emenecker, and Alex S. Holehouse. Direct prediction of intrinsically disordered protein conformational properties from sequence. *Nature Methods*, 21(3):465–476, March 2024. ISSN 1548-7105. doi: 10.1038/s41592-023-02159-5.
6. Joe G. Greener. Differentiable simulation to develop molecular dynamics force fields for disordered proteins. *Chemical Science*, 15(13):4897–4909, 2024. doi: 10.1039/D3SC05230C.
7. Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, Robert Verkuil, Ori Kabeli, Yaniv Shmueli, Allan dos Santos Costa, Maryam Fazel-Zarandi, Tom Sercu, Salvatore Candido, and Alexander Rives. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637):1123–1130, March 2023. doi: 10.1126/science.ade2574.
8. Ryan K Krueger and Max Ward. Jax-rnafold: scalable differentiable folding. *Bioinformatics*, 41(5):btaf203, 2025.
9. Ryan J Emenecker, Daniel Griffith, and Alex S Holehouse. Metapredict V2: An update to metapredict, a fast, accurate, and easy-to-use predictor of consensus disorder and structure. *BioRxiv*, pages 2022–06, 2022.
10. Kyosuke Adachi and Kyogo Kawaguchi. Predicting Heteropolymer Interactions: Demixing and Hypermixing of Disordered Protein Sequences. *Physical Review X*, 14(3):031011, July 2024. doi: 10.1103/PhysRevX.14.031011.
11. Garrett M Ginell, Ryan J Emenecker, Jeffrey M Lotthammer, Alex T Keeley, Stephen P Plassmeyer, Nicholas Razo, Emery T Usher, Jaqueline F Pelham, and Alex S Holehouse. Sequence-based prediction of intermolecular interactions driven by disordered regions. *Science*, 388(6749):eadq8381, 2025.