Supporting Information for

Learning to learn by using non-equilibrium training protocols for adaptable materials

Martin J. Falk, Jiayi Wu, Ayanna Matthews, Vedant Sachdeva, Nidhi Pashine, Marget L. Gardel, Sidney R. Nagel, Arvind Murugan

Arvind Murugan
E-mail: amurugan@uchicago.edu

This PDF file includes:

Supporting text
Figs. S1 to S7
SI References

Supporting Information Text

1. Designing adaptable allostery with gradient descent

We use a simple 22-node hexagonal lattice with boundary nodes fixed in space. With the geometry and rest lengths (which we define to be 1) fixed for all springs, we have 83 spring constants K = $\{k_i$, i =,1...,83$\}$. To construct the cost function, we first construct the dynamic matrix. By requiring the eigenvector of the lowest eigenstate for the dynamic matrix to be the structure we want, we get the cost function in (1). To make sure that the lowest energy state is well-separated from the rest of the states, we use a soft wall function $-\arctan(\frac{\lambda_2-\lambda_1}{\gamma})$, with λ_1 the lowest eigenvalue and λ_2 the second lowest eigenvalue, γ a parameter to make sure the gap between this two modes is large enough. The total function we are optimizing is given in equation (1) where $\vec{v}_{tar}$ is the desired or target pattern in the lowest eigenvector and $\vec{v}_0$ is the pattern in current lowest eigenvector.

$$f(\vec{v_0}) = max(\vec{v}_0 - \vec{v}_{tar}, 0) - \arctan\left(\frac{\lambda_2-\lambda_1}{\gamma}\right) \tag{1}$$

The initial condition for 83 spring constants is randomly generated from $U[0, 1]$ and we repeated the simulation 500 times. The optimizer we use is gradient descent with a learning rate $\alpha = 0.1$. To make sure the allostery response (the magnitude of the values in the resulting lowest eigenvector) is strong enough, we set the threshold of elongation along the direction of the spring for each node to be greater than or equal to 0.1 and the threshold of elongation perpendicular to the spring to less than or equal to 0.05. We set $\gamma = 0.05$. For a single goal, we use a max iteration time of 1000 steps. When we oscillate goals, we use the oscillating time periods of 200, 100 and 50 steps with maximum iteration time of 10000 steps. Every task will be terminated once converged.

2. Training for adaptable Poisson's ratios with local rules

A. Local Poisson's Ratio Training. In contrast to our allosteric training, we used disordered elastic networks when training for Poisson's ratio (ν). To construct these networks we used the PyCuda Packing Package(1, 2) from Simons Glass to make sets of jammed packings of 200 and 100 soft (harmonic) disks. We do this by initializing the pyCudaPacking Packing function, passing the number of particles and number of dimensions as 2. Particles are initially set at random positions by using the setRandomPositions function. Radii sizes are then set using the setLogNormalRadii function, which we pass the polydispersity of the packing as 0.1. We then set the packing density with the setPhi function, which we pass the density of 1. Finally, we energy minimize the packing using the minimizeFire function, setting the critical force parameter to 1e-20. From the packing, we create networks by setting the node positions of the network to the center points of each disk, which we get by saving the packing positions. The bonds between nodes are made where disks overlap, which we get using the getContacts function and requiring it only gives us stable contacts by passing the argument stable as true. We save the radii of the particles to get the stiffness of each bond, which we calculate as $\frac{1}{(r_i+r_j)^2}$, where r_i and r_j are the radii of the two disks that constitute the bond.

Next we need to apply bulk stresses to the networks, for which we use the rigidpy library(3). We first construct a rigidpy framework by passing the positions and bond list from the packing-generated network. To apply a shear, we construct a deformation matrix equal to $\begin{pmatrix} 1-\epsilon & 0 \\ 0 & \frac{1}{1-\epsilon} \end{pmatrix}$. To apply a compression, we construct a deformation matrix equal to $\begin{pmatrix} 1-\epsilon & 0 \\ 0 & 1-\epsilon \end{pmatrix}$. For both shear and compression, we set $\epsilon = 10^{-6}$.

To proceed with a single step of training, we generate a copy of the unstressed network and then deform it. The nodes are moved affinely according to the deformation matrix, and then relaxed with the rigidpy energyMinimizationNewton function with default parameters. We then use the rigidpy ElasticModulus function to calculate the elastic moduli given the above deformation matrices and compute the Poisson's ratio in the deformed network.

To take a training step and modify the network, we calculate bond stresses in the deformed network, and then select the bond with the absolute maximum stress. In the unstressed copy of the network we remove that bond by setting the spring constant to 0, thereby completing one step of the training.

B. Training Protocol Descriptions. For oscillatory training, we initially begin by cutting bonds that are the most stressed (absolute value of stress) under a shear compression, which decreases the Poisson's ratio ν. We calculate bond stresses by $\sigma_{ij} = kij(\ell_{ij} - \ell_{ij}^{(0)})$, where kij is the stiffness of the bond connecting nodes i and j, ℓ_{ij} is the length of that bond and $\ell_{ij}^{(0)}$ is the rest length. We remove N_c bonds in this way, before switching to removing the most stressed bond under bulk compression, which decreases ν. This proceeds for another N_c bond removals, before we go back to shear compression. We repeat each cycle of shear and bulk removals until the end of training.

For each oscillation frequency (200 nodes: $N_c = [20, 16, 10]$, 100 nodes: $N_c = [14, 11, 7]$), we perform 500 trajectories of oscillatory training. We needed to avoid incorporating networks which mechanically failed into our analysis, where we defined mechanical failure when the bulk modulus B = 0, indicating a possible crack in the network. We remove these from the analysis, because further training on them is impossible since there is no longer force transmission in the system. This is a consequence of removing bonds without replacement.

After running adaptable control protocols, we need to construct an ensemble of adaptable network pairs from these training trajectories. Of the trajectories which did not exhibit mechanical failure, we filtered those which reached both $\nu > 0.75$ and

$\nu < -0.75$ in their training trajectory and labeled these as successfully adaptable network pairs. These pairs are selected as the networks generated at the beginning of the last cycle, and the network at the end of the last cycle.

In order to understand our adaptable network pairs, we needed a control ensemble of network pairs trained for just a single target Poisson's ratio. We first generated single-target training trajectories with 500 trajectories each for both positive and negative target Poisson's ratios. These are trained separately for a positive and negative Poisson's ratio, starting from the same initial condition.

Having generated single-target training trajectories, we need to construct an ensemble of control network pairs. In order to make a fair comparison to the adaptably trained networks, we wanted to match the mean amount of time that single-goal networks were trained once they hit the target ν to the analogous quantity in the adaptable networks. Therefore, in the adaptive training protocol with 200 node networks ($N_c = 20$), we computed τ_{mean}, the average number of training steps after reaching the target ν for each trajectory that does not mechanically fail. Returning to our single-goal training trajectories, we chose a uniformly-distributed time point after reaching the target ν, with an average τ_{mean}. The network at this time point is selected as a control network, and we can generate pairs of control networks for both positive and negative ν.We ensured both of these successfully reached Poisson's ratios > 0.75 for the positive control and < -0.75 for the negative control without mechanical failure.

3. Designing adaptable heteropolymer folding with CMA-ES

Oscillatory training for adaptability in heteropolymer folding involves two ingredients: 1. a Langevin dynamics simulation implemented in the hoomd-blue software(4); and, 2. a covariance matrix adaptation evolution strategy (CMA-ES) implemented through the python package pycma(5).

A. Simulation Details. Our simulation consists of a single polymer confined within a box. The polymer is 13 monomers long, with each monomer being 1 unit length in diameter. The monomers are held together with harmonic springs of spring constant 100 kT, and additionally experience a harmonic bending force of 5 kT, giving them a persistence length roughly half the polymer length. All monomers experience a WCA pair-potential to enforce excluded volume. The crucial tunable monomer-monomer interactions are implemented through a short-range Morse potential:

$$V_{ij}(r) = \begin{cases} D_{ij}\left(\exp(-2\alpha(r-r_0)) - 2\exp(-\alpha(r-r_0)\right) & r < r_{cut} \\ 0 & r > r_{cut} \end{cases} \qquad [2]$$

We fix $\alpha = 6$, $r_0 = 1.1$, and $r_{cut} = 2$, so the potential minimum is centered close to the particle surface and is narrow. The pairwise affinities D_{ij} are set by the CMA-ES optimization procedure. Our optimization procedure acts on the affinities D_{ij}, which can be represented as a symmetric matrix. Since there is only one polymer in each simulation, we can safely set terms of the form D_{ii}, $D_{i+1,i}$, and $D_{i,i+1} = 0$. For a polymer with $N = 13$, this leaves us with $\frac{1}{2}(N \times N - N - 2(N-1)) = 66$ tunable D_{ij} to set through our optimization, which we describe in the following section.

Finally, the confining 2D square box is implemented through wall potentials which have a WCA form. Sides are of length 28. In all simulations unless otherwise noted, the polymer is initialized in the middle of the box as a straight line oriented along the x-axis. Positions are advanced with Langevin dynamics, with a drag coefficient of 5 and $kT = 1$. Timesteps are 5×10^{-3}, and we simulate all trajectories for 500 units of simulation time.

B. Optimization Details. Our optimization course can be broken down into epochs, which can be further broken down into rounds. For oscillatory training, one course of optimization is 10 epochs, each of which consists of 5 rounds. For the non-oscillatory training, there are two decoupled courses of training, each 1 epoch long, with each epoch being 25 rounds. At the end of each epoch, the target goal for the optimization switches, as described in the main text (Fig. 3A).

Each training course begins with an initial guess of $D_{ij} = 0$. Based on this guess, a population of 20 affinities matrices are sampled. Each member of the population has its fitness evaluated by undergoing 40 replicate simulations of the type described in the previous section. Each of these replicate simulations is run for 10^5 timesteps, at which point the final configuration obtained by the polymer is recorded. This ensemble of configurations is then used to evaluate the fitness of that member of the population. Based on the fitnesses of the current 20-member population ensemble, CMA-ES chooses a new 20-member population to evaluate, completing the optimization round. Additionally, at each round we record the affinity matrix of the population member that achieved the maximum fitness.

To complete our description of the training procedure, we describe the fitness function used in our implementation of CMA-ES. In compact notation, we can write this as:

$$f(w) = \min(Y, \frac{1}{N_{rep}} \sum_{i=1}^{N_{rep}} \Theta(w_i - W)) \qquad [3]$$

where N_{rep} is the number of simulation replicates, Θ is the Heaviside function, and w_i is the winding number of the polymer around either the first or last monomer, depending on what epoch the optimization is taking place in. The winding number w_i is evaluated in the final frame of each 500 unit time simulation. The yield cutoff Y and the winding threshold W are set to .7 and 1, respectively. During even-numbered epochs, the winding number is evaluated with respect to the last monomer, while in odd-numbered epochs the winding number is evaluated with respect to the first monomer. Qualitatively, this cost function computes the fraction of simulation replicates where the final configuration of the polymer has exceeded a winding number of 1 around the appropriate monomer. This is capped at a value of 70%, so there is no fitness gain above that threshold yield.

We chose the yield cutoff of 70% because this is the yield at which a polymer with an affinity matrix corresponding only to contacts formed in a close-packed hexagonal spiral performs (Supplemental Fig. 3). We assessed this by running 200 simulation runs of length 500 unit time, where the affinity matrix was non-zero only for contacts that appeared in the close-packed hexagonal spiral. Of these, 143 simulation runs exceeded the winding number threshold in their final frame.

C. Training Output Analysis.

Identifying high-yield matrices. In order to create ensembles of high-yield matrices, we run 200 optimization courses of oscillatory training, and 50 optimization courses of non-oscillatory training. For the oscillatory optimization courses, we monitor the maximum fitness value among the 20-member population sampled by CMA-ES in each round. If the maximum fitness value meets the .7 cap in the last three rounds of consecutive epochs, we record the pair of affinity matrices that achieved this high yield. In this way, we can construct an ensemble of matrix pairs identified in oscillatory training. This approach gives 62 matrix pairs in our ensemble. For the non-oscillatory optimization courses, we employ an identical procedure of selecting pairs in consecutive courses, which yields 40 matrix pairs in our ensemble.

Analyzing average affinity matrices. In Fig. 4C,D , we analyze the ensemble of affinity matrix pairs generated through oscillatory and non-oscillatory training. In Fig. 4C(bottom), we plot the mean affinity value in the upper triangular matrix, while in the lower triangle we indicate which matrix elements are in the upper quartile of mean affinity values. In Fig. 4C(top), we instead compute the difference between matrix pairs, and then average this across all matrix pairs. This quantity is displayed thresholded by 1kT. Note that all matrices are shuffled in terms of their residue numbering, so the elements corresponding to the ends of the polymer actually sit on at the center of the matrix. This is done for ease of comparison between the oscillatory and non-oscillatory training conditions. Finally in Fig. 4D, we compute the fraction of matrix elements which differ within a matrix pair. A matrix element is defined to be different if it exceeds a threshold of 1kT. This yields a distribution of fractions over our ensemble of matrix pairs, which is then displayed in violinplot format.

Analyzing kinetic effects. In Fig. 5, our goal is to identify the kinetic features of self-assembly that allow for adaptable heteropolymer folding. Specifically, we want to investigate the folding energy landscape as a function of the winding number of the folded part(s) of the spiral configuration. In order to do, we make the dramatic assumption that folding of the spiral or anti-spiral proceeds sequentially from one end of the polymer to the other. This assumption is clearly constraining, but not unreasonable. In two dimensions, successfully folding a tightly-wound spiral requires that, for any arbitrary radius of spiral, all the material that will eventually end up inside that radius must already be contained there. Once the curve closes, the excluded volume interactions between monomers would prevent the inclusion of any additional material. Applying this reasoning inductively implies that spiral formation must proceed sequentially from end to end. We note that this reasoning would not be true in three dimensions.

Under this assumption, we can estimate the folding energy landscapes generated for each affinity matrix pair. First, we record a final configuration achieved by one of the affinity matrices during a successful folding trajectory. This allows us to estimate the "on-target" folding energy landscape by sequentially laying down monomers in this configuration and recording the sum of the attractive and bending energies. We start from the monomers at the center of the spiral, and count any monomers within a distance of 1.4 simulation units to be in contact. Any monomer that is in contact under this definition get the maximum amount of attractive energy, corresponding to the depth of the well of the Morse potential used in simulation. Second, we record a final configuration achieved by the other matrix in the pair, which was trained to fold into the opposite spiral. We then repeat the sequential laying down process in order to estimate an "off-target" folding energy landscape. By adopting the convention that winding numbers for the anti-spiral are negative, we can construct energy landscapes as a function of winding number. We can then extract structural features of these landscapes, including the on-target fully-folded energy, the off-target fully-folded energy, as well as any nucleation barriers that exist along the way to these configurations.

The results of computing the off-target energy distributions is shown in Fig. 5B. In Fig. 5C, we display quantities related to the nucleation barriers of these energy landscapes. For every energy landscape, we use the scipy.find_peaks function to locate all peaks with a prominence exceeding 1kT, en route to folding the off-target configuration. We assign the last such peak to be the final barrier against folding the off-target structure. Having made this identification, we can query the off-target configuration to figure out which monomer-monomer contacts are involved in forming this final nucleation barrier against the off-target structure. Repeating this procedure across all energy landscapes, we collect a distribution of adjacency matrices which describe the off-target nucleation barrier configurations. We can use this to calculate the probability that a particular monomer-monomer interaction will appear in off-target nucleation barriers. To identify the most common such monomer-monomer interactions, we can threshold the distribution of these probabilities by the upper quartile of all non-zero probabilities. In Fig. 5C, we plot the resulting matrix elements on top of data from Fig. 4C.

4. Supplemental Figures

References

1. P Charbonneau, EI Corwin, G Parisi, F Zamponi, Universal microstructure and mechanical stability of jammed packings. *Phys. review letters* **109**, 205501 (2012).
2. PK Morse, EI Corwin, Geometric signatures of jamming in the mechanical vacuum. *Phys. review letters* **112**, 115701 (2014).
3. VF Hagh, M Sadjadi, rigidpy: Rigidity analysis in python. *Comput. Phys. Commun.* **275**, 108306 (2022).
4. JA Anderson, J Glaser, SC Glotzer, Hoomd-blue: A python package for high-performance molecular dynamics and hard particle monte carlo simulations. *Comput. Mater. Sci.* **173**, 109363 (2020).
5. N Hansen, Y Akimoto, P Baudis, Cma-es/pycma: r3. 0.3 (2020).

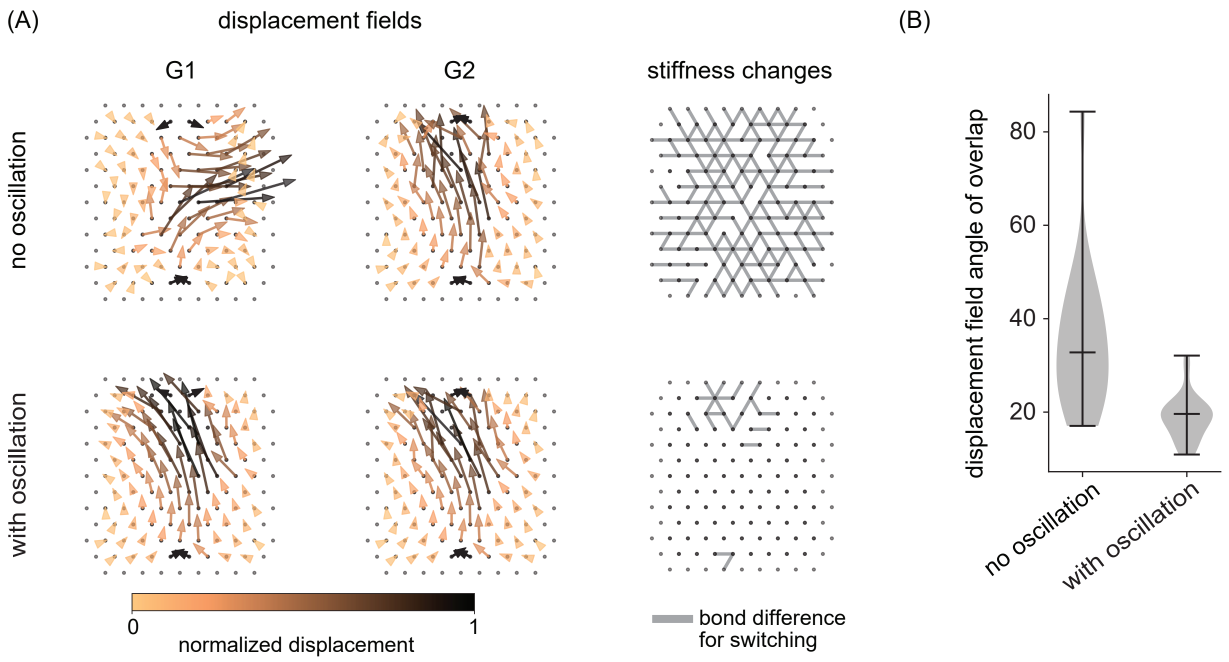


Fig. S1. Oscillatory training for adaptability creates coherent motions which are easily shifted to accomplish incompatible goals. (A) We train larger networks under both non-oscillatory (top) and oscillatory (bottom) conditions, and visualize the displacements of nodes in the lowest energy mode, as well as changes in relative bond stiffness exceeding .1; relative bond stiffness is defined by normalizing all bond stiffnesses to the mean stiffness of a network. We find that in all instances of training, the soft mode exhibits coherent motions. The same coherent motion is utilized to achieve both types of allostery in the oscillatory condition, with just a few bond stiffness changes necessary to switch from out-of-phase to in-phase motion. In contrast, two different coherent motions are selected in the non-oscillatory condition, with correspondingly large number of stiffness changes necessary for switching. (B) We calculate the absolute value of the overlap between lowest energy modes in networks designed to exhibit either in-phase or out-of-phase allosteric motions. We find that the distribution of the angle of this overlap (i.e. $\arccos|d_1 \cdot d_2|$, where d_i are the normalized displacement fields) is much smaller for networks trained with oscillation (n=50). Lines indicate min, median, and max of distribution.

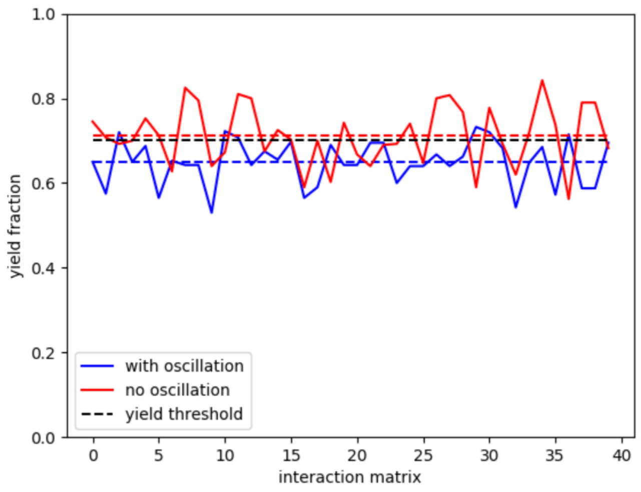
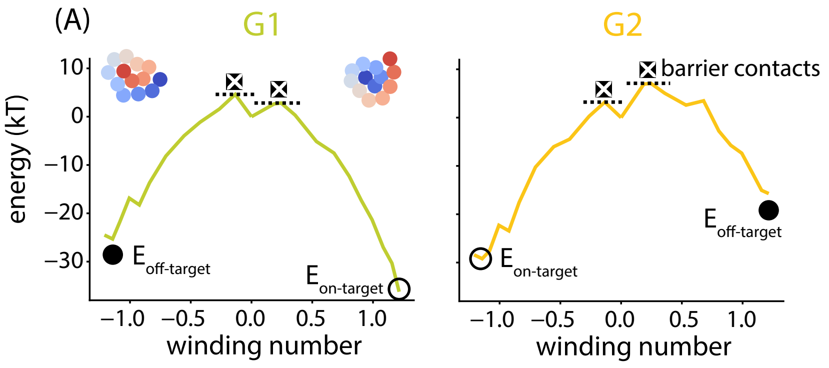


Fig. S2. The folding ability of adaptable affinity matrices does not significantly degrade relative to matrices trained for a single goal. During optimization, we estimate the yield of a given affinity matrix by computing 40 different independent simulation trajectories. Finite sample size effects could therefore potentially mask bi-stability in the resulting affinity matrices. In order to rule this out, we selected 40 fully-trained adaptable affinity matrices and 40 fully-trained single-goal affinity matrices, and simulated these matrices for 400 trajectories each.



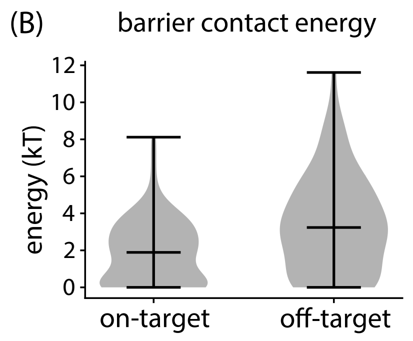
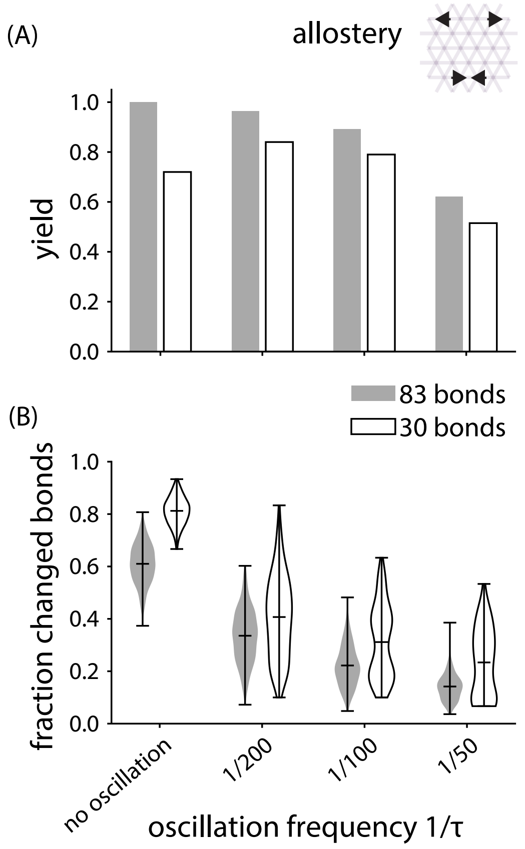


Fig. S3. Extracting structural features of folding from estimated energy landscapes. A. We construct estimated energy landscapes generated by each affinity matrix by sequentially laying down monomers into configurations drawn from successfully folded simulation, as described in Sec. C and with example configurations shown in the panel inset. By abusing notation and making the convention that winding around the red/blue end of the polymer is negative/positive winding, we can display energy landscapes as a function of a single winding number axis. We can repeat this procedure both for affinity matrices trained to accomplish G1 and for affinity matrices trained to accomplish G2 with oscillatory training. We plot a representative pair of curves for the G1 and G2 landscapes. To translate the visual language of the cartoon for adaptable energy landscapes in Fig. 5A(right), we mark off-target minima with black solid circles, and on-target energies with black open circles. Kinetic barriers to folding either of those states are indicated with white cross black squares. We find that the energy landscape pair displays the essential features of adaptable selection; deep energy minima on both on-target and off-target states, and kinetic energy barriers that are lower when passing to the on-target state from the initial zero-winding number configuration. (B) Across the ensemble of adaptable energy landscape pairs (n=62), we find that kinetic energy barrier (here, computed as simply the maximum energy along the pathway) to folding on-target configurations is lower than off-target configurations (~ 1.5kT). Lines indicate min, mean, and max of distribution.



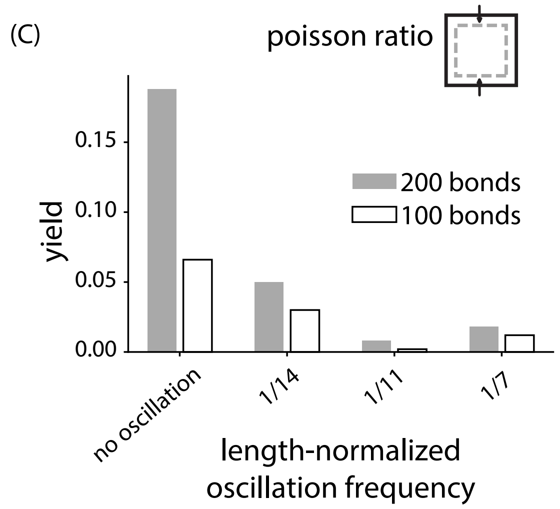


Fig. S4. Smaller systems are more difficult to train. (A) We performed training on the same system as discussed in main text Fig. 2, but in smaller elastic networks with only 30 adjustable bonds as opposed to 83. Yields of adaptable training have a decreasing trend, with lower yields compared to similar training in the 83-bond system considered in the main text (83 bonds, n=200; 30 bonds, n=200). (B) Fraction of bonds which need to be changed when switching between incompatible allosteric tasks in elastic networks. Decreasing fraction indicates that networks become more adaptable as the frequency of training oscillation increases (83 bonds, n=200; 30 bonds, n=200). τ is length of each training episode, as shown in main text Fig. 2C. (C) We performed training on the same system as discussed in main text Fig. 3, but in smaller networks with only 100 nodes as opposed to 200. Yields of adaptable training have a decreasing trend, with lower yields compared to similar training in the 200-node system considered in the main text. As yield is affected by the formation of system-spanning cracks, we compare oscillation frequencies in the 100-node networks to oscillation frequencies increased by a factor of $\sqrt{2}$ in the 200-node networks. This length-normalization allows us to control for the fact that it takes $\sqrt{2}$ times fewer bond removals to generate a crack in the 100-node networks compared to the 200-node networks; the area of the 100-node networks are reduced by a factor of 2 compared to the 200-node networks. (200 nodes, n=500; 100 nodes, n=500)

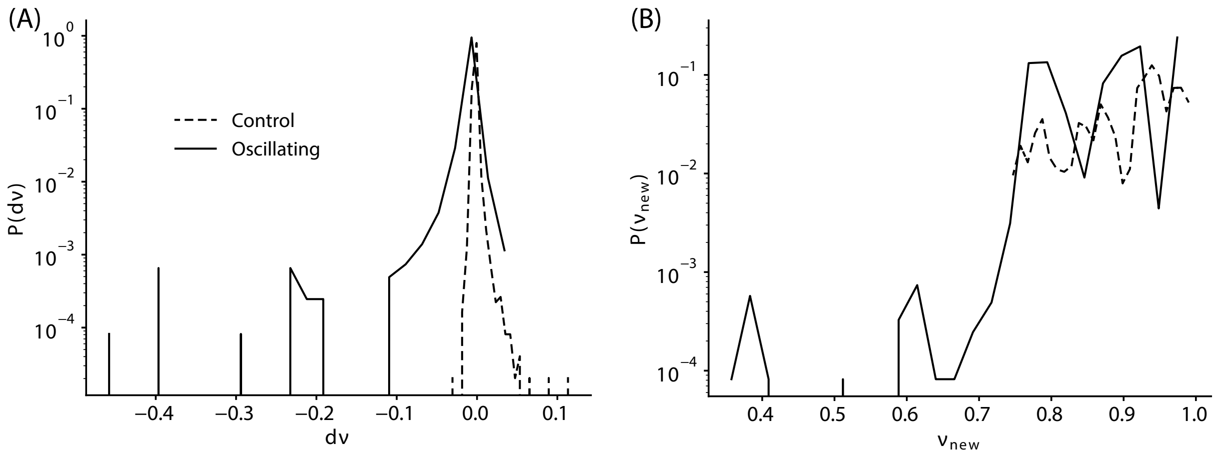


Fig. S5. Adaptable elastic networks are more sensitive to perturbations but overall have robust performance. We assessed the robustness of elastic networks trained for targeted Poisson's ratio (main text Fig. 3) by observing the effect of clipping any single bond in fully trained networks. Oscillating training was performed with a period of 20 bond removals. (A) Distribution of changes in Poisson's ratio (dν) computed by pruning a single bond. (B) Distribution of new Poisson's ratios (ν_{new}) following removal of a single bond (oscillating: n = 12209 bonds; control: n = 49373 bonds).

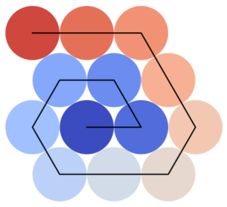


Fig. S6. A close-packed hexagonal configuration is used for yield calibration. By placing particles down on a hexagonal lattice, we create a compact polymer configuration with maximal winding number without stretching any harmonic bonds between adjacent monomers. We can use the adjacency matrix of this configuration to determine a maximal yield we should reasonably expect for our folding goal.

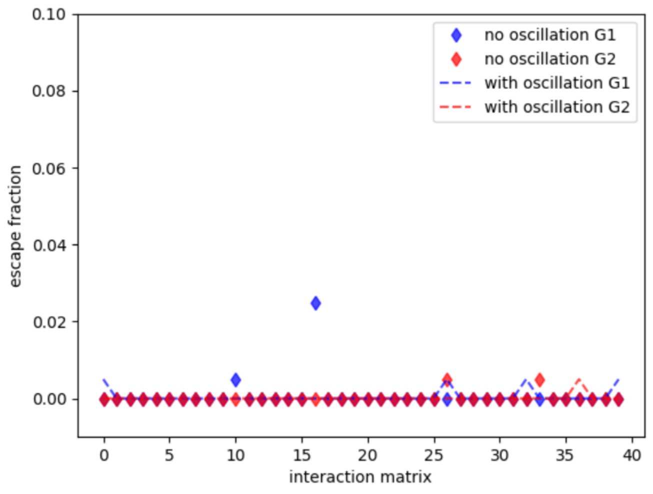


Fig. S7. Folding is kinetically controlled for our fully-trained affinity matrices. We select 40 fully-trained affinity matrix pairs from both the no-oscillation and with-oscillation training conditions. For each matrix pair, we initialize 400 simulations with the polymer positioned in its fully-folded configuration, and run simulations for the length of time we would run them to evaluate yield during optimization. We plot the fraction of simulations which record any unfolding event during the course of the simulation.