Optimizing protein fitness using Gibbs sampling with Graph-based Smoothing

Andrew Kirjner [*1] **Jason Yim** [*1] **Raman Samusevich** [2] **Tommi Jaakkola** [1] **Regina Barzilay** [1] **Ila Fiete** [1]

Abstract

The ability to design novel proteins with higher fitness on a given task would be revolutionary for many fields of medicine. However, brute-force search through the combinatorially large space of sequences is infeasible. Prior methods constrain search to a small mutational radius from a reference sequence, but such heuristics drastically limit the design space. Our work seeks to remove the restriction on mutational distance while enabling efficient exploration. We propose **G**ibbs sampling with **G**raph-based **S**moothing (GGS) which iteratively applies Gibbs with gradients to propose advantageous mutations using graph-based smoothing to remove noisy gradients that lead to false positives. Our method is state-of-the-art in discovering high-fitness proteins with up to 8 mutations from the training set. We study the GFP and AAV design problems, ablations, and baselines to elucidate the results. Code: https://github.com/kirjner/GGS

1. Introduction

In protein design, fitness is loosely defined as performance on a desired property or function. Examples of fitness include catalytic activity for enzymes (Anderson et al., 2021; Mazurenko et al., 2019) and fluorescence for biomarkers (Remington, 2011). Protein engineering seeks to design proteins with high fitness by altering the underlying sequences of amino acids. However, the number of possible proteins increases exponentially with sequence length, rendering it infeasible to perform brute-force search to engineer novel functions which often requires many mutations (i.e. at least 3 (Ghafari & Weissman, 2019)). Directed evolution (Arnold, 1998) has been successful in improving protein fitness, but it requires substantial labor and time to gradually explore many mutations.

Our work proposes a sequence-based method that can optimize over a noisy fitness landscape and efficiently sample large mutational edits. We introduce two methodological advances summarized in Figure 1. The first is Graph-based Smoothing (GS) that regularizes the noisy landscape. We formulate the landscape as a noisy graph signal and apply L_1 graph Laplacian regularization. This encourages sparsity and local consistency (Zhou et al., 2003) in the landscape: most protein sequences have zero fitness, and similar sequences have similar fitness. The effect is a smooth fitness landscape learned by the ML model on which gradients accurately approximate the direction towards high-fitness sequences. We utilize the improved gradients to reach high-fitness sequences requiring many mutations, by combining Gibbs With Gradients (GWG) sampling (Grathwohl et al., 2021) and a directed evolution-based procedure we call Iterative Extrapolation (IE), which applies multiple rounds of sampling over clustered sequences. Local improvements from the gradients on the smoothed landscape help select beneficial mutations to guide low-fitness sequences towards higher fitness, while multiple iterations of sampling allows exploration across many mutations.

We find GS and GWG sampling with IE are synergistic in enabling long-range exploration while avoiding the pitfalls of a noisy fitness landscape; the combination of both is referred to as GGS. We introduce a set of tasks using the Green Fluorescent Proteins (GFP) dataset (Sarkisyan et al., 2016) to simulate challenging protein design scenarios by starting with low-fitness sequences that require many (5 or more) mutations to the best fitness. We primarily study GFP because of (1) its difficulty as one of the longest proteins in fitness datasets and (2) its comprehensive fitness measurements of up to 15 mutations. To assess the generalizability of our method, we additionally study the Adeno-Associated Virus (AAV) dataset (Bryant et al., 2021) based on gene delivery fitness. We evaluate GGS and prior works on our proposed benchmarks to show that GGS is state-of-the-art in GFP and AAV fitness optimization.

2. Method

We begin with the problem formulation in Section 2.1. Our method utilizes Gibbs With Gradients (Section 2.2) sam-

[*]Equal contribution [1]Massachusetts Institute of Technology [2]IOCB, Czech Academy of Sciences, CIIRC, Czech Technical University in Prague. Correspondence to: Andrew Kirjner <kirjner@mit.edu>, Jason Yim <jyim@mit.edu>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

pling over a fitness landscape that has been smoothed with graph-based smoothing (Section 2.3). It then uses iterative extrapolation (Section 2.4) as a way to progressively extrapolate towards novel sequences. The full algorithm, Gibbs with Graph-based Smoothing (GGS), is presented in Algorithm 1.

2.1. Problem formulation

Let the starting set of length L protein sequences and their fitness measurements be denoted as $\mathcal{D}_0 = (\mathcal{X}_0, \mathcal{Y}_0)$ where $\mathcal{X}_0 \subset \mathcal{V}^L$ with vocabulary $\mathcal{V} = \{1, \dots, 20\}$ and $\mathcal{Y}_0 \subset \mathbb{R}$. We use subscripts to distinguish sequences, $x_i \in \mathcal{V}^L$, while a paranthetical subscript denotes the token, $(x_i)_j \in \mathcal{V}$ where $j \in \{1, \dots, L\}$. Note that our method can readily be extended to other modalities, e.g. nucleic acids.

For *in-silico* evaluation, we denote the set of *all* known sequences and fitness measurements as $\mathcal{D}^* = (\mathcal{X}^*, \mathcal{Y}^*)$. We assume there exists a black-box function $g : \mathcal{V}^L \to \mathbb{R}$ such that $g(x^*) = y^*$, which is approximated by an oracle g_ϕ. In practice, the oracle is a model trained with weights ϕ to minimize prediction error on $\mathcal{D}^*$. The starting dataset only includes low fitness sequences and is a strict subset of the oracle dataset $\mathcal{D}_0 \subset \mathcal{D}^*$ to simulate fitness optimization scenarios. Given $\mathcal{D}_0$, our task is to generate a set of sequences $\hat{\mathcal{X}} = \{\hat{x}_i\}_{i=1}^{N_{\text{samples}}}$ with higher fitness than the starting set.

2.2. GWG: Gibbs With Gradients

To generate initial candidates, we apply Gibbs With Graduents (GWG) (Grathwohl et al., 2021) to the protein sequences in $\mathcal{D}_0$. In this section, we provide the background from Grathwohl et al. (2021) tailored to protein fitness optimization. We first train a fitness *predictor*, $f_\theta : \mathcal{V}^L \to \mathbb{R}$, using $\mathcal{D}_0$, which acts as the learned unnormalized probability (i.e. negative energy) from sequence to fitness. We use the Mean-Squared Error (MSE) loss to train the predictor which we parameterize as a deep neural network. When training, we found it beneficial to employ negative data augmentation since both the dataset and the range of fitness values are small. Specifically, we double the size of the dataset by sampling random sequences, $x_i^{\text{neg}} \sim \text{Uniform}(\mathcal{V}^L)$, and assigning them the lowest possible fitness value, μ.

Our goal is to sample from $\log p(x) = f_\theta(x) - \log Z$ where Z is the normalization constant. Higher fitness sequences will be more likely under this distribution while sampling over many mutations will induce diversity and novelty. GWG uses Gibbs sampling with gradient-based approximations of *locally informed proposals* (Zanella, 2020):

$$q^\nabla(x'|x) \propto e^{\frac{(x')^\top d_\theta(x)}{2}} \mathbb{1}(x' \in H(x)),$$
$$d_\theta(x)_{ij} = \nabla_x f_\theta(x)_{ij} - x_i^T \nabla f_\theta(x)_i, \tag{1}$$

where $d_\theta(x)_{ij}$ is a first order Taylor approximation of the log-likelihood ratio of mutating the ith index of x to token j. Treating x and x' as one-hot, $(x')^\top d_\theta(x) = \sum_i (x_i')^\top d_\theta(x)_i$ is the sum over the local differences where x' differs from x. The proposal $q(x'|x)$ can be efficiently computed when $H(\cdot)$ is the 1-Hamming ball[1]: a single backward pass is needed to compute the Jacobian in Equation (1). A proposal mutation includes a sequence location, i^{loc}, and substitution, j^{sub}, that is sampled from a categorical distribution over the $L \times |\mathcal{V}|$ possibilities:

$$(i^{\text{loc}}, j^{\text{sub}}) \sim q(\cdot|x) = \text{Cat}\left(\text{Softmax}\left(\left\{\frac{d_\theta(x)_{i,j}}{\tau}\right\}_{(i,j)=(1,1)}^{(L,|\mathcal{V}|)}\right)\right) \tag{2}$$

where τ is a temperature hyperparameter. The proposal sequence x' is constructed by setting $x_{i^{\text{loc}}}$ to $\mathcal{V}_{j^{\text{sub}}}$. Each proposed sequence is accepted or rejected using Metropolis-Hastings (MH)

$$\min\left(\exp(f_\theta(x') - f_\theta(x)) \frac{q(i^{\text{loc}}, j^{\text{sub}}|x')}{q(i^{\text{loc}}, j^{\text{sub}}|x)},\ 1\right). \tag{3}$$

To summarize, our application of GWG first constructs N_{prop} sequences by sampling from Equation (2) then returns a set of accepted sequences, $\mathcal{X}'$, according to Equation (3). The full algorithm is provided in Algorithm 2.

Grathwohl et al. (2021) note that in GWG, the quality of the approximation in Equation (1) relies on the *smoothness* of the log-probability function (Theorem 1 in (Grathwohl et al., 2021)), which in our case is f_θ. We next describe a novel graph-based smoothing scheme used to satisfy this criterion in Section 2.3.

2.3. GS: Graph-based Smoothing

Given a predictor, $f_\theta : \mathcal{V}^L \to \mathbb{R}$, gradient-based methods for mutating towards high fitness sequences depend on the smoothness of the learned sequence-to-fitness mapping. Unfortunately, the high-dimensional sequence space coupled with few data points and noisy labels results in a noisy predictor that is prone to sampling false positives (Madry et al., 2017) or getting stuck in local optima (Brookes et al., 2019). To address this, we use techniques from graph signal processing to smooth the learned mapping by promoting similar sequences to have similar fitness (Zhou et al., 2003) while penalizing noisy predictions (Lu et al., 2016).

Suppose we have trained a noisy predictor with weights θ_0 on the initial dataset $\mathcal{D}_0$. To construct our graph $G = (V, E)$, we first construct the nodes V by iteratively applying pointwise mutations to each sequence in the initial set $\mathcal{X}_0$ to simulate a local landscape around each se-

[1] Defined as a ball using the Hamming distance.

quence. We call this routine `Perturb` with a hyperparameter N_{perturb} for the number of perturbations per sequence (see Algorithm 5). The edges, E, are a nearest neighbor graph with N_{neigh} neighbors where edge weights are inversely proportional to their Levenshtein distance, $\omega_{ij} = \omega((v_i, v_j)) = 1/\texttt{dist}(v_i, v_j)$; edge weights are stored in a similarity matrix $W = \{\omega_{ij}\ \forall v_i, v_j \in V\}$.

The normalized Laplacian matrix of G is $\mathcal{L} = I - D^{-1/2}WD^{-1/2}$ where I is the identity and D is a diagonal matrix with i-th diagonal element $D_{ii} = \sum_j \omega_{ij}$. An eigendecomposition of $\mathcal{L}$ gives $\mathcal{L} = U\Sigma U^T$ where Σ is a diagonal matrix with sorted eigenvalues along the diagonal and U is a matrix of corresponding eigenvectors along the columns. An equivalent eigendecomposition with symmetric matrix B is

$$\mathcal{L} = (\Sigma^{1/2}U^T)^T\Sigma^{1/2}U^T = B^TB, \qquad B = \Sigma^{1/2}U^T.$$

Next, we formulate smoothing as an optimization problem. For each node, we predict its fitness $\mathcal{S} = \{f_{\theta_0}(v)\ \forall v \in V\}$, also called the graph *signal*, which we assume to have noisy values. Our goal is to solve the following where $\mathcal{S}$ is arranged as a vector and $\mathcal{S}^*$ is the smoothed signal,

$$\mathcal{S}^* = \arg\min_{\hat{\mathcal{S}}} \|B\hat{\mathcal{S}}\|_1 + \gamma\|\hat{\mathcal{S}} - \mathcal{S}\|_1 \tag{4}$$

Equation (4) is a form of graph Laplacian regularization that has been studied for image segmentation with weak labels (Lu et al., 2016). B has eigenvalue weighted eigenvectors as rows. Due to the L_1-norm $\|B\hat{\mathcal{S}}\|_1$ is small if $\hat{\mathcal{S}}$ is primarily aligned with slowly varying eigenvectors whose eigenvalues are small. This term penalizes large jumps in fitness between neighboring nodes hence we call it *smoothness sparsity constraint*. The second term, $\|\hat{\mathcal{S}} - \mathcal{S}\|_1$, is the *signal sparsity constraint* that remove noisy predictions with hyperparameter γ. The L_1-norm is applied to reflect that most sequences have zero fitness.

At a high level, Equation (4) is solved by introducing auxiliary variables which allows for an approximate solution by solving multiple LASSO regularization problems (Tibshirani, 1996). Technical details and algorithm are described in Appendix C. Once we have $\mathcal{S}^*$, we retrain our predictor with the smoothed dataset $\mathcal{D} = (V, \mathcal{S}^*)$ on which the learned predictor is smoother with gradients much more amenable for gradient-based sampling, which we describe in the following section. We refer to our smoothing algorithm as Graph-based Smoothing (GS).

Even with a smoothed fitness landscape, GWG alone can only generate candidates that are a single mutation away from sequences in $\mathcal{D}_0$. The next section (Section 2.4) focuses on the development of an iterative framework to combine with GWG in order to improve the approximations for sequences that are multiple mutations away from the parent sequence.

2.4. IE: Iterative Extrapolation

The 1st order Taylor approximation of Equation (1) deteriorates the more we mutate from the parent sequence. Inspired by directed evolution (Arnold, 1998), we propose to alleviate this by performing multiple rounds of sampling where successive rounds use sequences from the previous round. Each round re-centers the Taylor approximation and extrapolates from the previous round. We first train a predictor f_θ using GS (Section 2.3). Prior to sampling, we observe the number of sequences may be large and redundant. To reduce the number of sequences, we perform hierarchical clustering (Müllner, 2011) and take the sequence of each cluster with the highest fitness using f_θ. Let $\mathcal{C}$ be the number of clusters.

$$\texttt{Reduce}(\ \{\mathcal{X}^c\}_{c=1}^{\mathcal{C}}; \theta) = \bigcup_{c=1}^{\mathcal{C}} \{\arg\max_{x \in \mathcal{X}^c} f_\theta(x)\}$$
$$\text{where } \{\mathcal{X}^c\}_{c=1}^{\mathcal{C}} = \texttt{Cluster}(\mathcal{X}; \mathcal{C}).$$

Each round r reduces the sequences from the previous round and performs GWG sampling.

$$\mathcal{X}'_{r+1} = \bigcup_{x \in \tilde{\mathcal{X}}_r} \texttt{GWG}(x; \theta), \quad \tilde{\mathcal{X}}_r = \texttt{Reduce}(\{\mathcal{X}_r^c\}_{c=1}^{\mathcal{C}}; \theta),$$
$$\{\mathcal{X}_r^c\}_{c=1}^{\mathcal{C}} = \texttt{Cluster}(\mathcal{X}'_r; \mathcal{C}).$$

One cycle of clustering, reducing, and sampling is a round of extrapolation,

$$\mathcal{X}'_{r+1} = \texttt{Extrapolate}(\mathcal{X}'_r; \theta, \mathcal{C}) \tag{5}$$

where the initial round $r = 0$ starts with $\mathcal{X}'_0 = \mathcal{X}_0$. After R rounds, we select our candidate sequences by taking the Top-N_{samples} sequences based on ranking with f_θ. We call this procedure Iterative Extrapolation (IE). While IE is related to previous directed evolution methods (Sinai et al., 2020), it differs by taking larger mutational edits on each round with GWG and encouraging diversity by mutating the best sequence of each cluster. The full candidate generation, Gibbs with Graph-based Smoothing (GGS), with IE is presented in Algorithm 1.

3. Experiments

We study the performance of GGS on optimizing fluorescence of Green Fluorescent Proteins (GFP) (Sarkisyan et al., 2016). We construct three difficulties of optimizing GFP based on the starting set of sequences. "Easy" difficulty is taken from a prior benchmark (Trabucco et al., 2021) which requires few mutations to achieve 99th percentile fitness while "medium" and "hard" requires 6 or more mutations. Description of each difficulty and analysis of the GFP dataset is provided in Appendix B. We train a oracle on the full dataset for *in-silico* evaluation of median fitness, diversity, and novelty. Training details, hyperparameters, descriptions of our metrics, and baselines are

Table 1. GFP optimization results (our oracle).

GFP Task		Method						
Difficulty	Metric	GFN-AL	CbAS	AdaLead	BO-qei	CoMs	PEX	**GGS**
Easy	Fit.	0.31 (0.1)	0.81 (0.0)	0.92 (0.0)	0.77 (0.0)	0.06 (0.3)	0.72 (0.0)	**0.93 (0.0)**
	Div.	19.1 (2.8)	4.5 (0.4)	2.0 (0.0)	5.9 (0.0)	129 (16)	2.0 (0.0)	2.0 (0.0)
	Nov.	215 (1.4)	1.4 (0.5)	1.0 (0.0)	0.0 (0.0)	164 (80)	1.0 (0.0)	1.0 (0.0)
Medium	Fit.	0.21 (0.1)	0.21 (0.0)	0.50 (0.0)	0.17 (0.0)	-0.1 (0.0)	0.50 (0.0)	**0.90 (0.0)**
	Div.	20.1 (4.2)	9.2 (1.5)	8.7 (0.1)	20.1 (7.1)	142 (16)	2.0 (0.0)	2.7 (0.0)
	Nov.	214 (4.6)	7.0 (0.7)	1.0 (0.0)	0.0 (0.0)	190 (11)	1.0 (0.0)	5.4 (0.4)
Hard	Fit.	0.19 (0.1)	-0.08 (0.0)	-0.04 (0.0)	0.01 (0.0)	-0.1 (0.2)	-0.11 (0.0)	**0.81 (0.0)**
	Div.	31.6 (2.4)	98.7 (16)	10.6 (0.3)	84.0 (7.1)	140 (7.1)	2.0 (0.0)	2.6 (0.0)
	Nov.	217 (3.9)	46.2 (9.4)	1.0 (0.0)	0.0 (0.0)	198 (2.9)	1.0 (0.0)	7.0 (0.0)

provided in Appendix D. Related works is provided in Appendix F. To demonstrate generalization to other fitness and proteins, we benchmark GGS on gene delivery using the Adeno-Associated Virus (AAV) (Bryant et al., 2021) dataset in Appendix E. Section 3.1 compares the performance of GGS on GFP to a representative set of baselines while Appendix E performs ablations on components of GGS.

3.1. Results

All methods are evaluated on 128 generated candidates, as done in design-bench. We run 5 seeds and report the average metric across all seeds including the standard deviation in parentheses. Results using our GFP oracle are summarized in Table 1. Results using the DB oracle are in Appendix E.

GGS substantially outperforms other baselines on the medium and hard difficulties, consistently navigating the mutational to achieve high fitness, while maintaining diversity and novelty from the training set. The unique extrapolation capabilities of GGS on the hardest difficulty level warranted additional analysis, and we investigate this further in Appendix G.

Adalead overall performed second-best, matching the performance of GGS on the easy difficulty with PEX only slightly worse. Notably, both Adalead and PEX suffer from a low novelty on all settings. GFN-AL exhibits subpar performance across all difficulty levels. Its performance notably deteriorates on medium and hard difficulty levels, a trend common amongst all baselines. CbAS explores very far, making on average 46 mutations, resulting in poor fitness. BO-qei is unable to extrapolate beyond the training set, and CoMs presents instability, as indicated by their high standard deviations, and collapse.[2]

[2]CoMs managed to generate only between 7 and 65 unique sequences.

We further analyze the distribution of novelty and fitness among CbAS, Adalead, and our method, GGS, in Figure 5. Adalead tends to be conservative, while CbAS is excessively liberal. GGS, on the other hand, manages to find the middle ground, displaying high fitness in its samples while also effectively exploring across the mutational gap at each difficulty level.

4. Discussion

We presented GGS, a method for optimizing protein fitness by incorporating ideas from MCMC, graph Laplacian regularization, and directed evolution. We outlined a new benchmark on GFP that introduces the challenge of starting with poor-fitness sequences requiring many edits to achieve high fitness. GGS discovered higher fitness sequences than in the starting set, even in the hard difficulty of our benchmark where prior methods struggled. We analyzed the two methodological advancements, Graph-based Smoothing (GS) and combining GWG with IE, as well as ablations to conclude each of these techniques aided GGS's performance.

There are multiple extensions of GGS. The first is to improve the application of GWG to proteins by removing the independence assumption across residues and instead modeling joint probabilities of epistatic interactions. One possibility for learning epistatic interactions is to incorporate 3D structure information (if available) to bias the sampling distribution. Secondly, the effectiveness of GS in our ablations warrants additional exploration into better regularization techniques for protein fitness predictors. Our formulation of GS is slow due to the nearest neighbor graph construction and its L_1 optimization. Lastly, investigating GGS to handle variable length sequences, multiple objectives, and multiple rounds of optimization is of high importance towards real protein engineering problems.

Acknowledgements

The authors thank Hannes Stärk, Rachel Wu, Nathaniel Bennett, Sean Murphy, Jaedong Hwang, Shahar Bracha, Josef Šivic, and Tomáš Pluskal for helpful discussion and feedback.

JY was supported in part by an NSF-GRFP. JY, RB, and TJ acknowledge support from NSF Expeditions grant (award 1918839: Collaborative Research: Understanding the World Through Code), Machine Learning for Pharmaceutical Discovery and Synthesis (MLPDS) consortium, the Abdul Latif Jameel Clinic for Machine Learning in Health, the DTRA Discovery of Medical Countermeasures Against New and Emerging (DOMANE) threats program, the DARPA Accelerated Molecular Discovery program and the Sanofi Computational Antibody Design grant. IF is supported by the Office of Naval Research, the Howard Hughes Medical Institute (HHMI), and NIH (NIMH-MH129046). RS was partly supported by the European Regional Development Fund under the project IMPACT (reg. no. CZ.02.1.01/0.0/0.0/15_003/0000468), the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (ID:90254), and the MISTI Global Seed Funds under the MIT-Czech Republic Seed Fund.

References

Anderson, D. W., Baier, F., Yang, G., and Tokuriki, N. The adaptive landscape of a metallo-enzyme is shaped by environment-dependent epistasis. *Nature Communications*, 12(1):3867, 2021.

Angermueller, C., Dohan, D., Belanger, D., Deshpande, R., Murphy, K., and Colwell, L. Model-based reinforcement learning for biological sequence design. 2020.

Arnold, F. H. Design by directed evolution. *Accounts of chemical research*, 31(3):125–131, 1998.

Arnold, F. H. Directed evolution: bringing new chemistry to life. *Angewandte Chemie International Edition*, 57(16): 4143–4148, 2018.

Biswas, S., Khimulya, G., Alley, E. C., Esvelt, K. M., and Church, G. M. Low-n protein engineering with data-efficient deep learning. *Nature methods*, 18(4):389–396, 2021.

Brookes, D., Park, H., and Listgarten, J. Conditioning by adaptive sampling for robust design. In *International conference on machine learning*, pp. 773–782. PMLR, 2019.

Bryant, D. H., Bashir, A., Sinai, S., Jain, N. K., Ogden, P. J., Riley, P. F., Church, G. M., Colwell, L. J., and Kelsic, E. D. Deep diversification of an aav capsid protein by machine learning. *Nature Biotechnology*, 39(6):691–696, 2021.

Dauparas, J., Anishchenko, I., Bennett, N., Bai, H., Ragotte, R. J., Milles, L. F., Wicky, B. I., Courbet, A., de Haas, R. J., Bethel, N., et al. Robust deep learning–based protein sequence design using proteinmpnn. *Science*, 378 (6615):49–56, 2022.

Ding, W., Nakai, K., and Gong, H. Protein design via deep learning. *Briefings in bioinformatics*, 23(3):bbac102, 2022.

Emami, P., Perreault, A., Law, J., Biagioni, D., and John, P. S. Plug & play directed evolution of proteins with gradient-based discrete mcmc. *Machine Learning: Science and Technology*, 4(2):025014, 2023.

Ghafari, M. and Weissman, D. B. The expected time to cross extended fitness plateaus. *Theoretical Population Biology*, 129:54–67, 2019. ISSN 0040-5809. doi: https://doi.org/10.1016/j.tpb.2019.03.008. URL https://www.sciencedirect.com/science/article/pii/S0040580918301011. Special issue in honor of Marcus Feldman's 75th birthday.

Grathwohl, W., Swersky, K., Hashemi, M., Duvenaud, D., and Maddison, C. Oops i took a gradient: Scalable sampling for discrete distributions. In *International Conference on Machine Learning*, pp. 3831–3841. PMLR, 2021.

Ingraham, J., Baranov, M., Costello, Z., Frappier, V., Ismail, A., Tie, S., Wang, W., Xue, V., Obermeyer, F., Beam, A., et al. Illuminating protein space with a programmable generative model. *bioRxiv*, pp. 2022–12, 2022.

Jain, M., Bengio, E., Hernandez-Garcia, A., Rector-Brooks, J., Dossou, B. F. P., Ekbote, C. A., Fu, J., Zhang, T., Kilgour, M., Zhang, D., Simine, L., Das, P., and Bengio, Y. Biological sequence design with GFlowNets. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9786–9801. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/jain22a.html.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Lee, M., Vecchietti, L. F., Jung, H., Ro, H., Cha, M., and Kim, H. M. Protein sequence design in a latent space via model-based reinforcement learning.

Lu, Z., Fu, Z., Xiang, T., Han, P., Wang, L., and Gao, X. Learning from weak and noisy labels for semantic segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(3):486–500, 2016.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Maus, N., Jones, H., Moore, J., Kusner, M. J., Bradshaw, J., and Gardner, J. Local latent space bayesian optimization over structured inputs. *Advances in Neural Information Processing Systems*, 35:34505–34518, 2022.

Mazurenko, S., Prokop, Z., and Damborsky, J. Machine learning in enzyme engineering. *ACS Catalysis*, 10(2): 1210–1223, 2019.

Meier, J., Rao, R., Verkuil, R., Liu, J., Sercu, T., and Rives, A. Language models enable zero-shot prediction of the effects of mutations on protein function. *Advances in Neural Information Processing Systems*, 34:29287–29303, 2021.

Müllner, D. Modern hierarchical, agglomerative clustering algorithms. *arXiv preprint arXiv:1109.2378*, 2011.

Notin, P., Dias, M., Frazer, J., Hurtado, J. M., Gomez, A. N., Marks, D., and Gal, Y. Tranception: protein fitness prediction with autoregressive transformers and inference-time retrieval. In *International Conference on Machine Learning*, pp. 16990–17017. PMLR, 2022.

Padmakumar, V., Pang, R. Y., He, H., and Parikh, A. P. Extrapolative controlled sequence generation via iterative refinement. *arXiv preprint arXiv:2303.04562*, 2023.

Rao, R., Bhattacharya, N., Thomas, N., Duan, Y., Chen, X., Canny, J., Abbeel, P., and Song, Y. S. Evaluating protein transfer learning with tape. In *Advances in Neural Information Processing Systems*, 2019.

Remington, S. J. Green fluorescent protein: a perspective. *Protein Science*, 20(9):1509–1519, 2011.

Ren, Z., Li, J., Ding, F., Zhou, Y., Ma, J., and Peng, J. Proximal exploration for model-guided protein sequence design. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 18520–18536. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/ren22a.html.

Sarkisyan, K. S., Bolotin, D. A., Meer, M. V., Usmanova, D. R., Mishin, A. S., Sharonov, G. V., Ivankov, D. N., Bozhanova, N. G., Baranov, M. S., Soylemez, O., et al. Local fitness landscape of the green fluorescent protein. *Nature*, 533(7603):397–401, 2016.

Sinai, S., Wang, R., Whatley, A., Slocum, S., Locane, E., and Kelsic, E. D. Adalead: A simple and robust adaptive greedy search algorithm for sequence design. *arXiv preprint arXiv:2010.02141*, 2020.

Stanton, S., Maddox, W., Gruver, N., Maffettone, P., Delaney, E., Greenside, P., and Wilson, A. G. Accelerating bayesian optimization for biological sequence design with denoising autoencoders. *arXiv preprint arXiv:2203.12742*, 2022.

Sun, H., Dai, H., Xia, W., and Ramamurthy, A. Path auxiliary proposal for mcmc in discrete space. In *International Conference on Learning Representations*, 2022.

Tibshirani, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

Trabucco, B., Kumar, A., Geng, X., and Levine, S. Conservative objective models for effective offline model-based optimization. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 10358–10368. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/trabucco21a.html.

Trabucco, B., Geng, X., Kumar, A., and Levine, S. Design-bench: Benchmarks for data-driven offline model-based optimization. *CoRR*, abs/2202.08450, 2022. URL https://arxiv.org/abs/2202.08450.

Trippe, B. L., Yim, J., Tischer, D., Broderick, T., Baker, D., Barzilay, R., and Jaakkola, T. Diffusion probabilistic modeling of protein backbones in 3d for the motif-scaffolding problem. *arXiv preprint arXiv:2206.04119*, 2022.

Watson, J. L., Juergens, D., Bennett, N. R., Trippe, B. L., Yim, J., Eisenach, H. E., Ahern, W., Borst, A. J., Ragotte, R. J., Milles, L. F., Wicky, B. I. M., Hanikel, N., Pellock, S. J., Courbet, A., Sheffler, W., Wang, J., Venkatesh, P., Sappington, I., Torres, S. V., Lauko, A., De Bortoli, V., Mathieu, E., Barzilay, R., Jaakkola, T. S., DiMaio, F., Baek, M., and Baker, D. Broadly applicable and accurate protein design by integrating structure prediction networks and diffusion generative models. *bioRxiv*, 2022.

Wilson, J. T., Moriconi, R., Hutter, F., and Deisenroth, M. P. The reparameterization trick for acquisition functions. *arXiv preprint arXiv:1712.00424*, 2017.

Yim, J., Trippe, B. L., De Bortoli, V., Mathieu, E., Doucet, A., Barzilay, R., and Jaakkola, T. Se (3) diffusion model with application to protein backbone generation. *arXiv preprint arXiv:2302.02277*, 2023.

Zanella, G. Informed proposals for local mcmc in discrete spaces. *Journal of the American Statistical Association*, 115(530):852–865, 2020.

Zhang, R., Liu, X., and Liu, Q. A langevin-like sampler for discrete distributions. *International Conference on Machine Learning*, 2022.

Zhou, D., Bousquet, O., Lal, T., Weston, J., and Schölkopf, B. Learning with local and global consistency. *Advances in neural information processing systems*, 16, 2003.

A. Figures and algorithms.

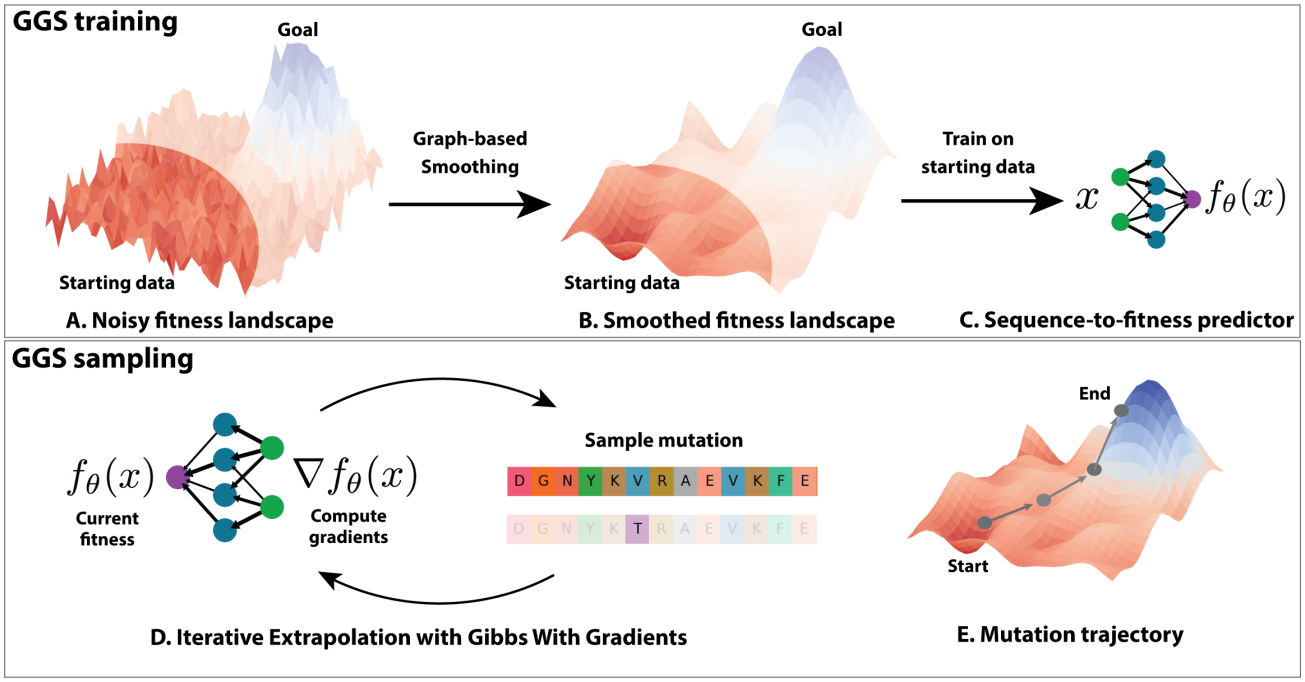

Figure 1. GGS overview. **(A)** Protein engineering is often challenging due to a noisy fitness landscape on which the starting dataset (unblurred) is a fraction of landscape with the highest fitness sequences hidden (blurred). **(B)** We develop Graph-based Smoothing (GS) to estimate a smoothed fitness landscape from the starting data. Intuitively, the gradients allow extrapolation towards higher fitness sequences. **(C)** A fitness predictor is trained on the smoothed fitness landscape. **(D)** Gradients from the fitness predictor are used in an iterative sampling procedure called Iterative Extrapolation (IE) where Gibbs With Gradients (GWG) is performed on each step with renewed gradient computations. **(E)** Each step of IE samples mutations towards higher fitness.

Algorithm 1 `GGS`: Gibbs sampling with Graph-based Smoothing

Require: Starting dataset: $\mathcal{D}_0 = (\mathcal{X}_0, \mathcal{Y}_0)$
Require: GWG hyperparameters: N_{prop}, τ, M
Require: GS hyperparameters: $N_{\text{neigh}}, N_{\text{perturb}}, \gamma$
Require: IE hyperparameters: $N_{\text{samples}}, R, \mathcal{C}$

Construct negative data.
$\mathcal{D} \leftarrow \mathcal{D}_0 \cup \{(x_i^{\text{neg}}, \mu)\}_{i=1}^{|\mathcal{D}_0|}$
Initial training.
$\theta_0 \leftarrow \arg\max_{\tilde{\theta}} \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[(y - f_{\tilde{\theta}}(x))^2\right]$
GS Algorithm 3.
$\theta \leftarrow \texttt{Smooth}(\mathcal{X}_0; \theta_0)$
Initial round of IE
$\{\mathcal{X}_0\}_{c=1}^{\mathcal{C}} \leftarrow \texttt{Cluster}(\mathcal{X}_0; \mathcal{C})$
$\tilde{\mathcal{X}}_0^c \leftarrow \texttt{Reduce}(\{\mathcal{X}_0\}_{c=1}^{\mathcal{C}}; \theta)$
$\mathcal{X}_0' \leftarrow \cup_{x \in \tilde{\mathcal{X}}_0^c} \texttt{GWG}(x; \theta)$
# GWG Algorithm 2
for $r = 1, \ldots, R$ **do**
  # Remaining rounds of IE Equation (5)
  $\mathcal{X}_r' \leftarrow \texttt{Extrapolate}(\mathcal{X}_{r-1}'; \theta)$
end for
# Return Top-N_{samples} sequences based on predicted fitness f_θ.
$\hat{\mathcal{X}} \leftarrow \texttt{TopK}(\cup_{r=1}^{R} \mathcal{X}_r')$
Return $\hat{\mathcal{X}}$

Algorithm 2 GWG: Gibbs With Gradients

Require: Parent sequence: x
Require: Predictor weights: θ
Require: Sampling temperature: τ
Require: Number of sequences to sample: N_{prop}
Require: Amino acid vocabulary: $\mathcal{V}$

$\mathcal{X}' \leftarrow \emptyset$
for $i = 1, \ldots, N_{\text{prop}}$ **do**
  # Enumerate number of Gibbs samples
  $x' \leftarrow x$
  # Sample index and token Equation (2)
  $(i^{\text{loc}}, j^{\text{sub}}) \sim q(\cdot|x)$
  # Construct proposed candidate
  $x'i^{\text{loc}} \leftarrow \mathcal{V}j^{\text{sub}}$
  if accept using Equation (3) **then**
    $\mathcal{X}' \leftarrow \mathcal{X}' \cup \{x'\}$
  end if
end for
# Return accepted sequences.
Return $\mathcal{X}'$

Algorithm 3 GS: Graph-based Smoothing

Require: Sequences: $\mathcal{X}$
Require: Predictor weights: θ_0
Require: Number of perturbations: N_{perturb}
Require: Number of neighbors: N_{neigh}
Require: Sparsity weight: γ

Construct graph nodes Algorithm 5.
$V \leftarrow \texttt{Perturb}(\mathcal{X}, N_{\text{perturb}})$
Construct similarity matrix.
$W \leftarrow \{\omega_{ij} = 1/\texttt{dist}(v_i, v_j) : v_i, v_j \in V\}$
N_{neigh} nearest neighbor graph edges based on W.
$E \leftarrow \texttt{NearestNeighbor}(\mathcal{X}; W, N_{\text{neigh}})$
Node attributes
$\mathcal{S} \leftarrow \{f_{\theta_0}(v)\ \forall v \in V\}$
Solves Equation (4) with Algorithm 4
$\mathcal{S}^* \leftarrow \texttt{NoiseReduction}(V, E, \mathcal{S}, \gamma)$
$\mathcal{D} \leftarrow (V, \mathcal{S}^*)$
Train on smoothed dataset.
$\theta \leftarrow \arg\max_{\tilde{\theta}} \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[(y - f_{\tilde{\theta}}(x))^2\right]$
Return θ

Algorithm 4 `NoiseReduction`: follows Algorithm 1 in Lu et al. (2016)

Require: Nodes: V
Require: Edges: E
Require: Noisy labels: $\mathcal{S}$
Require: Sparsity weight: γ
$\mathcal{L} \leftarrow$ Compute normalized Laplacian on graph $G = (V, E)$.
$V_E \leftarrow$ Find 50 smallest eigenvectors* of $\mathcal{L}$.
$\hat{\mathcal{S}} \leftarrow \mathcal{S}$
for $i = 1 \ldots 1000$ optimization rounds **do**
  $A^* \leftarrow$ Solution in Equation (9) using off-the-shelf solvers.
  $F^* \leftarrow U_V A^*$
  $\hat{\mathcal{S}}^* \leftarrow \texttt{SoftThreshold}(F^*, \mathcal{S}, \gamma)$
end for
Return $\hat{\mathcal{S}}^*$

Algorithm 5 `Perturb`

Require: Sequences: $\mathcal{X}$
Require: Number of perturbations: N_{perturb}
$V \leftarrow \mathcal{X}$
while $|V| < (|\mathcal{X}| \cdot N_{\text{perturb}})$ **do**
  $x \sim \text{Uniform}(V)$
  $x' \leftarrow$Random point mutation to x
  $V \leftarrow V \cup \{x'\}$
end while
Return V

B. Benchmark and dataset

We use the Green Fluoresent Protein (GFP) dataset from Sarkisyan et al. (2016) containing over 56,806 log fluorescent fitness measurements, with 51,715 unique amino-acid sequences due to *sequences having multiple measurements*. We quantify the difficulty of a protein fitness optimization task by introducing the concept of a *mutational gap*, which we define as the minimum Levenshtein distance between any sequence in the training set to any sequence in the 99th percentile:

$$\text{Gap}(\mathcal{X}_0; \mathcal{X}^{99\text{th}}) = \texttt{min}(\{\texttt{dist}(x, \tilde{x}) : x \in \mathcal{X}, \tilde{x} \in \mathcal{X}^{99\text{th}}\})$$

A mutational gap of 0 means that the training set, $\mathcal{D}_0$, may contain sequences that are in the 99th percentile of fitness. Solving such tasks is easy because methods may sample high-fitness sequences from the training set. Prior work commonly uses the GFP task introduced by design-bench (DB) evaluation framework (Trabucco et al., 2022) which has a mutational gap of 0 (discussed later). To compare to previous work, we include the DB task as **"easy"** difficulty in our experiments, but we introduce **"medium"** and **"hard"** optimization tasks which have lower starting fitness ranges in the 20-40th and 10-30th percentile of known fitness measurements alongside much higher mutational gaps. Our proposed difficulties are summarized in Table 2 and visualized in Figure 3.

Table 2. Proposed GFP tasks

Difficulty	Range (%)	$\lvert\mathcal{D}_0\rvert$	Gap
Medium	20th-40th	2828	6
Hard	10th-30th	1636	7

The oracle in design-bench (DB) uses a Transformer-based architecture from Rao et al. (2019). When using this oracle, we noticed a concerning degree of false positives and a thresholding effect of its predictions. We propose a simpler CNN architecture as the oracle that achieves superior performance in terms of Spearman correlation and fewer false positives

as seen in Figure 4. Our CNN consists of a 1D convolutional layer that takes in a one-hot encoded sequence, followed by max-pooling and a dense layer to a single node that outputs a scalar value. It uses 256 channels throughout for a total of 157,000 parameters – 15 fold fewer than DB oracle.

Our experiments in Section 3 benchmark on GFP easy, medium, and hard with our CNN oracle. In Appendix E we summarize an additional benchmark using Adeno-Associated Virus (AAV) dataset (Bryant et al., 2021) which focuses on optimizing a 28-amino acid segment for DNA delivery. We use the same task set-up and train our CNN oracle on AAV.

Design-bench difficulty. Prior works have used the GFP task introduced by design-bench (DB), a suite of model-based reinforcement learning tasks (Trabucco et al., 2022), which *samples* a starting set of 5,000 sequences from the 50-60th percentile fitness range. The wild-type (WT) sequences have 534 fluorescence measurements ranging from the 58th to the 100th percentile (Figure 2) which implies it may be in the training set depending on the random seed[3] and allow strong performance by memorization. Even if there is no data leakage (due to the random seed), the 50-60th percentile fitness starting range includes sequences that require 1 mutation to reach the 99th percentile (see Figure 3). The task is overly simplified in this sense and does not provide accurate evaluation of different methods.

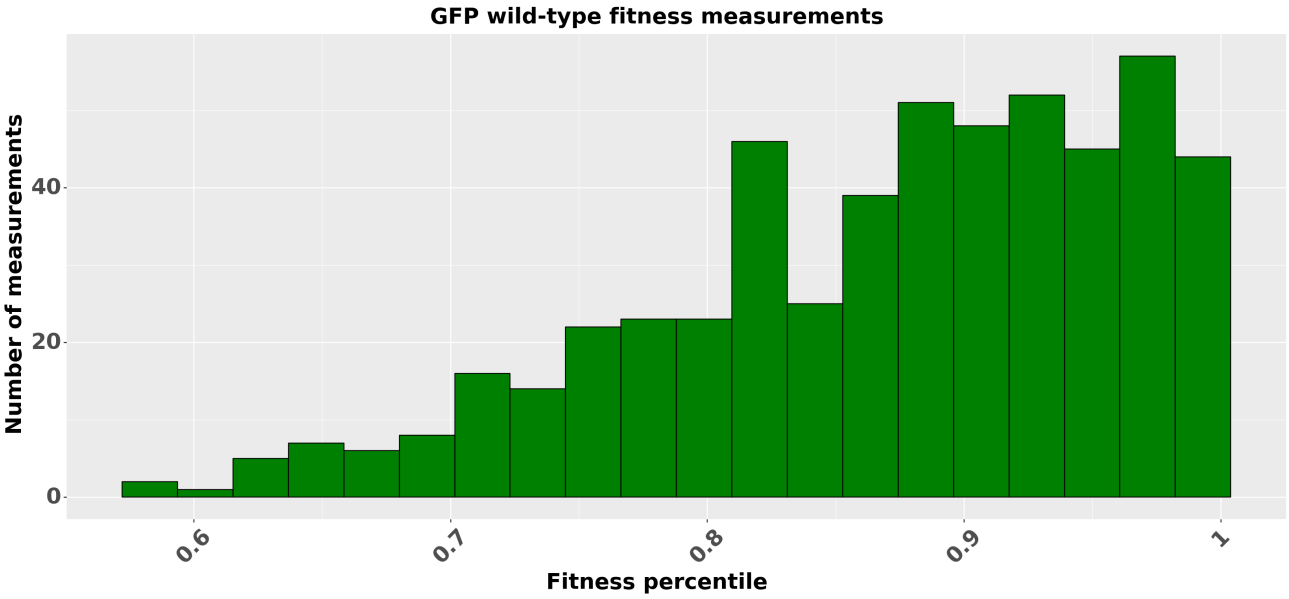

Figure 2. GFP wild-type multiplicity. The same wildtype sequence is measured a total of 534 times in the Sarkisyan et al. (2016) dataset with a wide range of fitness measurements from the 58th to 100 percentile. As a result, it is possible for the wild-type or other sequences with multiple measurements to contaminate the training set when filtering only on fitness range.

[3]We confirmed this to be the case in the design-bench codebase.

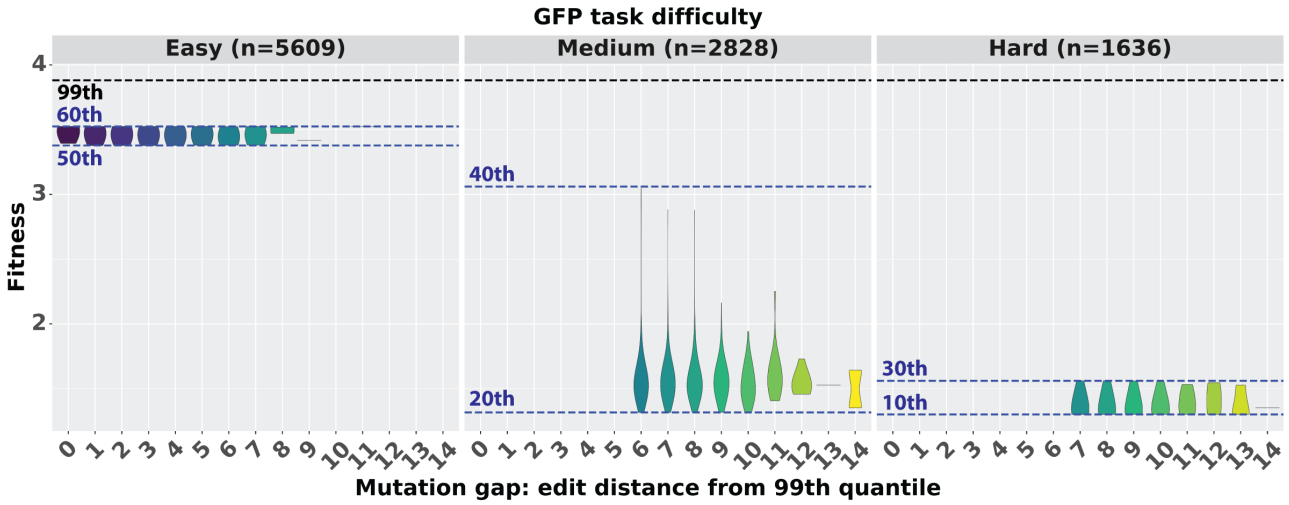

Figure 3. GFP task difficulty comparisons. **Easy difficulty** is taken from design-bench. We take all GFP sequences between the 50-60th percentile regardless of distance to sequences in the 99th percentile of GFP sequences resulting in 5609 sequences the method has access to. Data leakage is present due to the multiplicity of GFP measurements that allows the wild-type sequence and other top sequences to be present in the 50-60th percentile. **Medium difficulty** filters the starting dataset to have sequences in the 20-40th percentile and be 6 or more mutations away from anything in the top 99th percentile resulting in 2628 sequences in the starting dataset. **Hard difficulty** filters the starting dataset to have sequences in the 10-30th percentile and be 7 or more mutations away from anything in the top 99th percentile resulting in 1636 sequences.

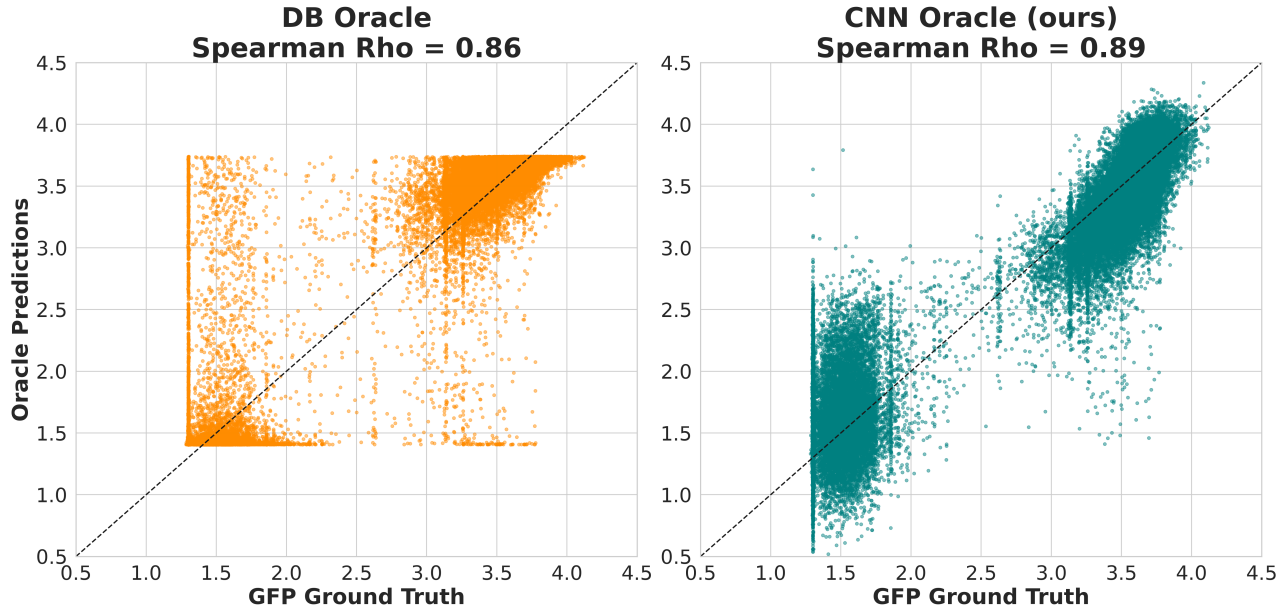

Figure 4. Design-bench (DB) and our CNN oracle Comparison. DB oracle exhibits strange behaviour of predictions being thresholded at min/max values. Yet the Spearman correlation (rho) is high and closely matches the reported rho in (Trabucco et al., 2021). Our simpler CNN oracle on the other hand is able to fit the ground truth data more accurately (higher rho) and has less false positives. We perform experiments with both oracles in our work.

C. Additional methods

In this section, we provide additional details of Graph-based Smoothing (GS) and algorithms. Algorithm 2 describes pseudo code for Gibbs With Gradients (GWG) sampling introduced in Section 2.2. Algorithm 3 describes pseudo code for GS described in Section 2.3 with Algorithm 4 and Algorithm 5 as sub-routines used in GS.

We now describe remaining details of GS. We use the same optimization algorithm from Lu et al. (2016) to solve Equation (4) which is reproduced here. As a reminder, Equation (4) is

$$\mathcal{S}^* = \arg\min_{\hat{\mathcal{S}}} \|B\hat{\mathcal{S}}\|_1 + \gamma\|\hat{\mathcal{S}} - \mathcal{S}\|_1.$$

Solving the above, a combination of L_1-optimization problems, is notoriously difficult therefore we introduce an auxiliary variable F with same dimensions as $\mathcal{S}$ and instead solve the following,

$$\min_{\hat{\mathcal{S}}\geq 0, U} \frac{1}{2}\|\hat{\mathcal{S}} - F\|_{\mathcal{F}}^2 + \lambda\|BF\|_1 + \gamma\|\hat{\mathcal{S}} - \mathcal{S}\|_1 \tag{6}$$

where $\|\cdot\|_{\mathcal{F}}$ is the Frobenius norm. The purpose of F is to disentangle $\hat{\mathcal{S}}$ from being part of two L_1-norm terms such that we can solve smaller sub-problems. The new term $\|\hat{\mathcal{S}} - F\|_{\mathcal{F}}^2$ enforces F to be close to $\hat{\mathcal{S}}$. Equation (6) can be solved with two easier sub-problems.

$$F^* = \arg\min_{F} \frac{1}{2}\|F - \hat{\mathcal{S}}^*\|_{\mathcal{F}}^2 + \lambda\|BF\|_1 \tag{7}$$

$$\hat{\mathcal{S}}^* = \arg\min_{\hat{\mathcal{S}}\geq 0} \frac{1}{2}\|\hat{\mathcal{S}} - F^*\|_{\mathcal{F}}^2 + \gamma\|\hat{\mathcal{S}} - \mathcal{S}\|_1 = \texttt{SoftThreshold}(F^*, \mathcal{S}, \gamma) \tag{8}$$

where $\hat{\mathcal{S}}^* = \mathcal{S}$ initially and λ is a Lagrange multiplier. Equation (8) has a closed-form solution using the soft-threshold function (`SoftThreshold`) which is defined in eq. (9) of Lu et al. (2016) (we omit it here for ease of exposition). Equation (7) requires iterative optimization due to the computational intractability of B. This is overcome by a dimensionality reduction,

$$F = U_V A, \quad U_V = [u_1|\ldots|u_{50}]$$

where U_V is a matrix whose columns are the 50 smallest eigenvectors of $\mathcal{L}$ and $A = \{a_{ij}\}$ is the reconstruction coefficients. Lu et al. (2016) let the number of projected eigenvectors be a hyperparameter but we found 50 to work well. Equation (7) can be reformulated as,

$$\begin{aligned} F^* &= \arg\min_{A} \frac{1}{2}\|U_V A - \hat{\mathcal{S}}^*\|_{\mathcal{F}}^2 + \lambda\|BU_V A\|_1 \\ &= \arg\min_{A} \sum_j \underbrace{\frac{1}{2}\|U_V A_{\cdot j} - \hat{\mathcal{S}}^*_{\cdot j}\|_{\mathcal{F}}^2 + \lambda\|BU_V A_{\cdot j}\|_1}_{(*)} \end{aligned} \tag{9}$$

where $\hat{\mathcal{S}}^*_{\cdot j}$ and $A_{\cdot j}$ denotes the j-th column of $\hat{\mathcal{S}}^*$ and A respectively. Lu et al. (2016) proved solving Equation (9) is equivalent to solving Equation (8) under certain conditions but is a good approximation otherwise. Each (*) can be solved independently,

$$\begin{aligned} (*) &= \arg\min_{A_{\cdot j}} \frac{1}{2}\|U_V A_{\cdot j} - \hat{\mathcal{S}}^*_{\cdot j}\|_2^2 + \lambda\|BU_V A_{\cdot j}\|_1 \\ &= \arg\min_{A_{\cdot j}} \frac{1}{2}\|U_V A_{\cdot j} - \hat{\mathcal{S}}^*_{\cdot j}\|_2^2 + \lambda\sum_i \Sigma_{ii}^{\frac{1}{2}}|a_{ij}| \end{aligned} \tag{10}$$

See Lu et al. (2016) for derivation. Recall Σ_{ii} is the ith eigenvalue. Equation (10) can be solved using off-the-shelf solvers. We can now solve Equation (7) and Equation (8) in iterative fashion. We set a number of rounds (1000 in our case) and alternate between solving Equation (8) and Equation (7). The full algorithm is provided as `NoiseReduction` in Algorithm 4 which follows Algorithm 1 of Lu et al. (2016).

D. Experiment details

GGS training and sampling. Following the oracle discussion in Appendix B, we use the oracle CNN architecture for our predictor (but trained on different data). To ensure a fair comparison, we use the same predictor across all model-based baselines. We use the following hyperparameters as input to Algorithm 1 across all tasks: $N_{\text{prop}} = 100$, $\tau = 0.01$, $M = 5$, $N_{\text{neigh}} = 500$, $N_{\text{perturb}} = 1000$ $N_{\text{samples}} = 128$ $R = 10$, $\mathcal{C} = 500$. We were unable to perform extensive exploration of hyperparameters. Reducing the number of hyperparameters and finding optimal values is an important future direction. Training is performed with batch size 1024, ADAM optimizer (Kingma & Ba, 2014) (with $\beta_1 = 0.9, \beta_2 = 0.999$), learning rate 0.0001, and 1000 epochs using a single A6000 Nvidia GPU. Initial predictor training takes 10 minutes while graph-based smoothing takes around 30 minutes depending on convergence of the numerical solvers. Training with the smoothed data takes 4 to 8 hours. Sampling takes under 30 minutes and can be parallelized.

Baselines. We choose a representative set of prior works with publicly available code: GFlowNets (GFN-AL) (Jain et al., 2022), model-based adaptive sampling (CbAS) (Brookes et al., 2019), greedy search (AdaLead) (Sinai et al., 2020), bayesian optimization with quasi-expected improvement acquisition function (BO-qei) (Wilson et al., 2017), conservative model-based optimization (CoMs) (Trabucco et al., 2021), and proximal exploration (PEX) (Ren et al., 2022).

Metrics. Each method generates $N_{\text{samples}} = 128$ samples $\hat{\mathcal{X}} = \{\hat{x}_i\}_{i=1}^{N_{\text{samples}}}$ to evaluate. Here, `dist` is the Levenshtein distance. We report three metrics:

- **(Normalized) Fitness** = $\texttt{median}(\{\xi(\hat{x}_i; \mathcal{Y}^*)\}_{i=1}^{N_{\text{samples}}})$ where $\xi(\hat{x}; \mathcal{Y}^*) = \frac{g_\phi(\hat{x}_i) - \texttt{min}(\mathcal{Y}^*)}{\texttt{max}(\mathcal{Y}^*) - \texttt{min}(\mathcal{Y}^*)}$ is the min-max normalized fitness.
- **Diversity** = $\texttt{mean}(\{\texttt{dist}(x, \tilde{x}) : x, \tilde{x} \in \hat{\mathcal{X}}, x \neq \tilde{x}\})$ is the average sample similarity.
- **Novelty** = $\texttt{median}(\{\eta(\hat{x}_i; \mathcal{X}_0)\}_{i=1}^{N_{\text{samples}}})$ where $\eta(x; \mathcal{X}_0) = \texttt{min}(\{\texttt{dist}(x, \tilde{x}) : \tilde{x} \in \mathcal{X}^*, \tilde{x} \neq x\})$ is the minimum distance of sample x to any of the starting sequences $\mathcal{X}_0$.

We use median for outlier robustness. Diversity and novelty were introduced in Jain et al. (2022). We emphasize that higher diversity and novelty is *not* equivalent to better performance. For instance, a random algorithm would achieve maximum diversity and novelty.

E. Additional results

AAV evaluation. Our experiments use GFP (Appendix B) due to its long sequence length (237 residues) resulting in a large search space and wide coverage of fitness measurements (56,806 data points). The proposed medium and hard difficulties were possible because of GFP's measurements up to 15 mutations. We sought to experiment on a different fitness landscape to see if (1) mutational gap is a valid criterion for task difficulty and (2) our method isn't over-optimized for fluorescence fitness in GFP.

A second commonly studied protein fitness dataset is the Adeno-Associated Virus (AAV) (Bryant et al., 2021). The fitness of the AAV capsid protein is its ability to package a DNA payload, i.e. gene delivery. This dataset encompasses 201,426 subsequences of the AAV2 wild-type known to be immportant for gene delivery based on the structure. Of these, we consider 44,156 subsequences of length 28 and consist solely of substitutions[4]. The benchmark difficulties on AAV are defined the same way in the GFP task. The same oracle architecture is used to train on all 44,156 sequences while GGS uses the same hyperparameters as in GFP.

Results from evaluating GGS and baselines on AAV is presented in Table 3. GGS outperforms all baselines on this dataset as well, demonstrating its robustness across another protein fitness landscapes. The improvement using GGS over baselines is lower for AAV compared to GFP. This could be explained due to the smaller search space (28 residues) than GFP (237 residues) suggesting AAV is easier. We observe a decrease in fitness performance across all methods to suggest mutational gap is a valid criterion for task difficulty.

Results with DB oracle. Table 4 shows results of using the design-bench (DB) oracle compared against our CNN oracle. We find GGS is still state-of-the-art on all fitness metrics.

[4]AAV contains sequences of varying length with insertions and deletions. We excluded these since our method and the majority of our baselines cannot handle insertion and deletions.

Table 3. AAV optimization results (our oracle).

AAV Task		Method						
Difficulty	Metric	GFN-AL	CbAS	Adalead	BO-qei	CoMs	PEX	**GGS**
Easy	Fit.	-0.01 (0.0)	0.53 (0.0)	0.53 (0.0)	0.42 (0.2)	0.31 (0.1)	0.45 (0.0)	**0.62 (0.0)**
	Div.	11.8 (1.7)	6.7 (0.6)	3.7 (0.1)	10.1 (7.2)	4.3 (1.71)	2.1 (0.1)	2.7 (0.0)
	Nov.	20.4 (0.8)	5.6 (0.6)	1.0 (0.0)	8.4 (7.6)	5.7 (0.6)	1.0 (0.0)	1.0 (0.0)
Medium	Fit.	0.03 (0.0)	0.35 (0.0)	0.46 (0.0)	0.30 (0.1)	0.32 (0.1)	0.34 (0.0)	**0.62 (0.0)**
	Div.	14.6 (1.6)	13.7 (0.4)	7.9 (0.1)	17.3 (3.6)	10.2 (3.7)	2.4 (0.2)	2.5 (0.0)
	Nov.	21.4 (0.9)	7.8 (0.5)	1.0 (0.0)	4.2 (9.4)	9.0 (2.0)	1.1 (0.2)	5.0 (0.0)
Hard	Fit.	0.02 (0.0)	0.31 (0.0)	0.37 (0.0)	0.28 (0.0)	0.31 (0.2)	0.27 (0.0)	**0.43 (0.0)**
	Div.	17.8 (1.0)	15.1 (0.7)	11.1 (0.1)	20.8 (1.6)	9.2 (4.2)	2.0 (0.0)	8.3 (0.5)
	Nov.	20.8 (1.0)	8.4 (0.5)	1.0 (0.0)	0.0 (0.0)	8.2 (2.6)	1.0 (0.0)	8.0 (0.0)

Table 4. GFP optimization results with design-bench (DB) oracle. Note that diversity and novelty

GFP Task		Method						
Difficulty	Metric	GFN-AL	CbAS	AdaLead	BO-qei	CoMs	PEX	**GGS**
Easy	Fit.	0.05 (0.0)	0.84 (0.0)	**0.86 (0.0)**	0.84 (0.0)	0.05 (0.1)	0.71 (0.2)	**0.86 (0.0)**
	Div.	19.1 (2.8)	4.5 (0.4)	2.0 (0.0)	5.9 (0.0)	129 (16)	2.0 (0.0)	2.0 (0.0)
	Nov.	215 (1.4)	1.4 (0.5)	1.0 (0.0)	0.0 (0.0)	164 (80)	1.0 (0.0)	1.0 (0.0)
Med.	Fit.	0.05 (0.0)	0.0 (0.0)	0.56 (0.2)	0.0 (0.0)	0.05 (0.1)	0.60 (0.3)	**0.84 (0.1)**
	Div.	20.1 (4.2)	9.2 (1.5)	8.7 (0.1)	20.1 (7.1)	142 (16)	2.0 (0.0)	2.7 (0.0)
	Nov.	214 (4.6)	7.0 (0.7)	1.0 (0.0)	0.0 (0.0)	190 (11)	1.0 (0.0)	5.4 (0.4)
Hard	Fit.	0.05 (0.0)	0.0 (0.0)	0.05 (0.1)	0.0 (0.0)	0.05 (0.2)	0.0 (0.0)	**0.72 (0.3)**
	Div.	31.6 (2.4)	98.7 (16)	10.6 (0.3)	84.0 (7.1)	140 (7.1)	2.0 (0.0)	2.6 (0.0)
	Nov.	217 (3.9)	46.2 (9.4)	1.0 (0.0)	0.0 (0.0)	198 (2.9)	1.0 (0.0)	7.0 (0.0)

Ablations. We perform ablations on the two stages of GGS on the hard difficulty task. In the first ablation, we remove GS and start sampling after initial predictor training. The second ablation runs IE for fewer that 15 iterations (1, 5, and 10).

Our results are shown in Table 5. We see GS is crucial for GGS on the hard difficulty level. When IE is run for 1 or 5 iterations there is also a large decrease in performance. Running IE for 10 iterations achieves equivalent fitness, but slightly worse diversity and novelty. We conclude each component of GGS contributes to its performance.

Table 5. Ablation results (our oracle).

Difficulty	Metric	**GGS**	without GS	IE10	IE5	IE1
Hard	Fitness	**0.81 (0.0)**	-0.03 (0.0)	**0.81 (0.0)**	0.40 (0.0)	0.07 (0.0)
	Diversity	2.6 (0.0)	22.7 (1.6)	2.5 (0.0)	5.7 (0.0)	13.5 (0.0)
	Novelty	7.0 (0.0)	13.4 (0.5)	6.7 (0.4)	5.0 (0.0)	1.0 (0.0)

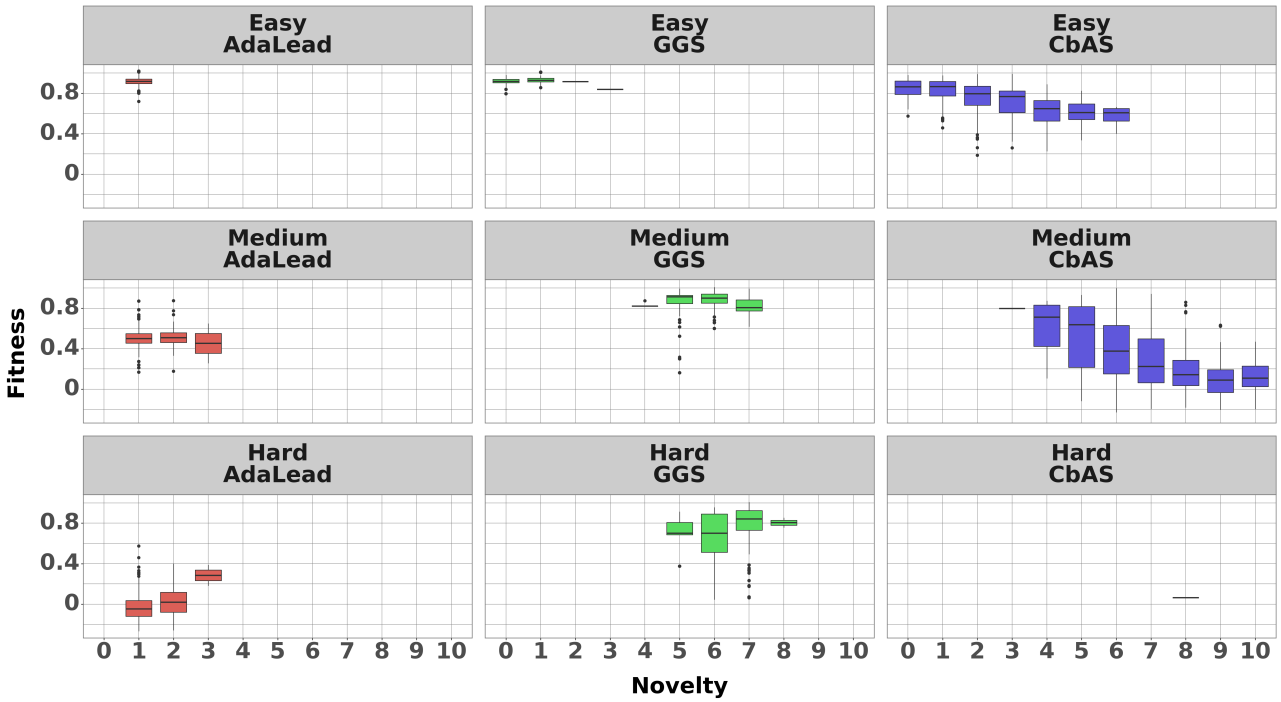

Figure 5. Comparison of GFP novelty and fitness on samples from AdaLead, GGS, and CbAS. From left to right, we observe increasing exploration behaviour from the respective methods. However, only GGS maintains high fitness while exploring the novel sequences. Nearly all samples from CbAS on hard are beyond 10 novelty and have very low fitness.

F. Related work

Optimization in protein design. Approaches in protein design can broadly be categorized in using sequence, structure or both (Ding et al., 2022). Advances in structure-based protein design have been driven by a combination of geometric deep learning and generative models (Watson et al., 2022; Ingraham et al., 2022; Yim et al., 2023; Dauparas et al., 2022; Trippe et al., 2022). Sequence-based protein design has been explored through the lens of reinforcement learning (Angermueller et al., 2020; Lee et al.), latent space optimization (Stanton et al., 2022; Lee et al.; Maus et al., 2022), GFlowNets (Jain et al., 2022), Bayesian optimization (Wilson et al., 2017), generative models (Brookes et al., 2019; Biswas et al., 2021; Notin et al., 2022; Meier et al., 2021), and model-based directed evolution (Sinai et al., 2020; Arnold, 2018; Padmakumar et al., 2023; Ren et al., 2022; Trabucco et al., 2021). Together they face the common issue of a noisy landscape to optimize over. Moreover, fitness labels are problem-dependent and scarce, apart from well-studied proteins (Biswas et al., 2021). Our method addresses small amounts of starting data and noisy landscape by regularization with GS. We focus on sequence-based methods where we use locally informed Markov Chain Monte Carlo (MCMC) (Zanella, 2020) with Gibbs With Gradients (GWG) (Grathwohl et al., 2021) which requires a smooth energy function for strong performance guarantees. Concurrently, Emami et al. (2023) used GWG to sample higher fitness sequences by optimizing over a product of experts distribution, a mixture of a protein language model and a fitness predictor. However, they eschewed the need for a smooth energy function which we address with GS.

Discrete MCMC. High-dimensional discrete MCMC can be inefficient with slow mixing times. GWG showed discrete MCMC becomes practical by utilizing learned gradients in the sampling distribution, but GWG in its published form was limited to sampling in a proposal window of size 1. Zhang et al. (2022) proposed to modify GWG with Langevin dynamics to allow for the whole sequence to mutate on every step while Sun et al. (2022) augmented GWG with a path auxiliary proposal distribution to propose a series of local moves before accepting or rejecting. We find that GGS, which combines GWG with IE is simpler and effective in achieving a proposal window size beyond 1 by using multiple iterations.

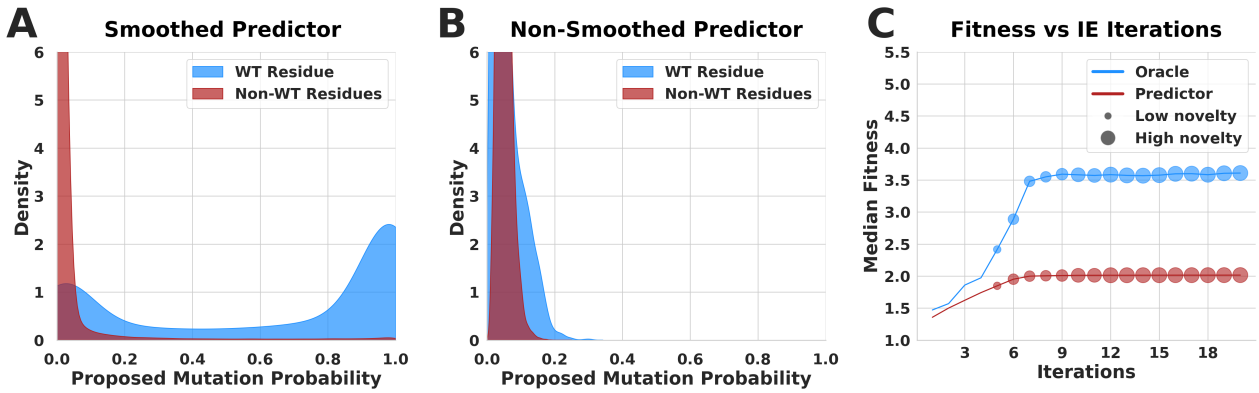

Figure 6. Analysis of GGS for GFP hard Task. **(A, B)** Proposed mutation probability of WT residue vs. non-WT residues for subsequently accepted mutations with and without GS. The non-smoothed predictor gives the WT residue only slightly higher probability than other residues. **(C)** Un-normalized median fitness scores for the smoothed predictor and CNN oracle vs. number of IE iterations. Larger points correspond to higher median novelty, with the smallest point having median novelty 5. Median fitness increases until about 7 iterations of IE, while median novelty increases until 12 iterations of IE.

G. Analysis

We analyze GGS to understand its ability in sampling higher fitness sequences. Focusing on the hard GFP task, we investigated (1) if GS results in higher probability towards higher fitness mutations and (2) how GWG with IE produces sequences with improved fitness across large mutational distances over multiple iterations.

To answer (1), we analyze the probability of the GFP wild-type (WT) which is representative of high-fitness sequences in the 99th percentile. Figure 6A plots the probability density function of sampling mutations on residues (line 4 of Algorithm 2) different from the WT vs. the background distribution of all other residues using GS. Comparing this with Figure 6B, which does not use GS, we observe GS is crucial for sampling towards high fitness sequences in the form of the WT.

Next, we consider how the performance and novelty of GGS change as the number of IE iterations increases. Figure 6C shows that the smoothed predictor and oracle fitness predictions of the same sequences both plateau around 7 iterations, while novelty peaks later. Running IE for a suitably large number of iterations is therefore necessary to achieve both high fitness and high novelty. Furthermore, the correlation between oracle and predictor performance suggests that using predicted fitness to choose the number of IE iterations is warranted in our GFP benchmark. We expect different behaviours for different proteins and fitnesses. Overall the use of smoothing and iterative refinement with IE is promising in our experiments.