Data-Driven Optimization for Protein Design: Workflows, Algorithms and Metrics

Anonymous authors
Paper under double-blind review

Abstract

Recent works have successfully demonstrated the ability of deep neural networks in predicting important properties such as fitness and stability from protein sequences via supervised learning. However, the use of learned deep neural network models for the task of designing *de novo* proteins that maximize a certain fitness value with backbones from scratch remains under-explored. In this paper, we study the problem of designing proteins where the optimization be carried out in a purely data-driven, "offline" manner, by utilizing databases of experimental data collected from wet lab evaluations. Synthesis of proteins proposed by the algorithm in an experimental setup in a wet lab, which incurs a big manual overhead for designers, is not allowed. Such an offline optimization problem require that a practitioner make several several design choices: a designer must decide what data distribution to train on, how their method would be evaluated, and must additionally devise workflows for tuning the optimization method they wish to use. In this paper, we perform a systematic study of various design choices that arise in in protein design, grounded in the problem of optimizing for protein stability, and use these insights to propose workflows, protocols and metrics to assist practitioners in effectively applying such data-driven approaches to protein design problems.

1 Introduction

Recent works demonstrate the ability for deep neural networks to learn informative features from protein sequences as demonstrated by performance on *supervised prediction* tasks, such as breakthroughs in structure prediction (Jumper et al., 2021; Baek et al., 2021). This ability of deep neural networks to pick up on generalizing patterns from databases of proteins presents an opportunity for algorithmic *design* in biology. One would expect that end-to-end learning based methods can learn to design novel protein sequences by leveraging complex features which are inaccessible to human domain experts, to increase success-to-cost ratios in experimentation and reduce costs and time. In this paper, we consider such a design task, where the goal is to propose desirable *inputs* to a sequence-to-function model, rather than simply producing output values of predictive success. Such techniques can unlock tremendous implications for the development of drugs and therapeutics. In existing works, this problem setting is referred to as model-guided exploration (Sinai and Kelsic, 2020) or ***model-based optimization*** **(MBO)** (Hie and Yang, 2022). Additionally, we focus on the *offline* setting, which is in contrast to the online setting where additional experimental iterations are necessary to improve the model or the designed inputs. Such techniques reduce the experimental cost and time necessary, and furthermore enable a purely data-driven process which enables practitioners without access to wet lab facilities to do protein design.

While some prior work have explored this setting (Bryant et al., 2021; Brookes et al., 2019; Trabucco et al., 2021b; Sinai et al., 2020), Such model-based optimization methods have been shown to have empirical wet-lab success (Bryant et al., 2021), and while some algorithmic decisions such as preventing the exploitation of inaccurately high-scoring regions of the model Brookes et al. (2019); Trabucco et al. (2021b); Kumar and Levine (2019); Fu and Levine (2021) or exploration strategies (Sinai et al., 2020) are effective in model-based optimization,

there remains a large number of design decisions which can greatly affect the usability of these methods for drug discovery. For instance, it remains unclear how to obtain a construct a training dataset that is most amenable to data-driven optimization, given a large amount of logged experimental data obtained through wet lab experiments; it is unclear how a practitioner should reliably evaluate the efficacy of the optimized sequences found by an ML method in the absence of actual wet lab experiments. In a setting where single mutations can cause the protein to lose its function, a more rigorous look into such decisions will enable practitioners to bring these techniques to more widespread use in application. Furthermore, publicly available data collected without machine learning in mind may represent very small portions of the search space and its local neighbors (e.g. sequences from single-site or multi-site mutagenesis), or fail to fit cleanly into a machine learning problem setting, leading to a brittle learned sequence-to-function mappings. In this paper, we aim to provide guidelines, workflows and metrics to assist practitioners in making such design decisions.

The main contribution of this work is a systematic study of various design choices that arise in offline data-driven optimization for protein design. In addition to several design factors associated with offline optimization algorithms such as model capacity, and regularization, the design factors we study also include the composition of the training dataset extracted from raw wet lab experimental data, computational methods for evaluating the efficacy of a designed protein sequence without any wet lab evaluation. We propose metrics and guidelines that will hopefully serve as a "user manual" for assisting biology practitioners in making such design decisions in a data-driven fashion, without needing wet lab evaluation. We evaluate our proposals, metrics and guidelines on the task of designing novel *de novo* peptides, and validate their efficacy under a variety of design scenarios that are typically of interest to a protein designer. We hope that these results will encourage practitioners to utilize data-driven offline optimization approaches for protein engineering.

2 Preliminaries

Offline model-based optimization. In this work, we examine the setting of *offline* model-based optimization. In the case of protein engineering, the offline setting contrasts with the online setting in that further experimental iterations can be performed to improve our model and proposed designs. This offline setting is appealing as iterative experimentation can be costly, and a purely data-driven design process is more accessible. In the setting of offline MBO, an algorithm is provided access to a static dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$ of designs $\mathbf{x}_i$ and a corresponding measurement of the objective value y_i. The algorithm consumes this dataset and produces an optimized candidate design $\mathbf{x}^*$ which is evaluated against the true objective function.

Offline MBO problem often involves learning a proxy objective f_θ mapping input sequence to measured property of interest, which we hereafter refer to as the **design model**. Optimization is then performed to find an input which maximizes this learned design model: $\mathbf{x}^* = \arg\max_{\mathbf{x}} f_\theta(\mathbf{x})$. In applying MBO to biological data, the workflow may resemble: (1) collect data (experimentally or from publicly available sources); (2) supervised learning of the model (or finetune a pre-existing pretrained model); (3) optionally add conservatism, if needed; (4) *in silico* directed evolution or other classes of discrete optimization methods; and (5) evaluation by multiple oracle models. We refer to the **oracle model** as a model which (ideally) approximates the ground truth for evaluating new designs, following prior work (Brookes et al., 2019; Trabucco et al., 2021a). In reality, these so-called "oracles" are often also imperfect. We outline further strategies for each of these workflow steps in Section 3.

Designing stable proteins. To empirically examine design decisions involved in the effectiveness of model-based optimization in protein engineering, we consider the task of designing stable *de novo* miniproteins. We examine stability as a holistically scored measurement. The dataset is from Rocklin et al. (2017), where sequences are derived from four idealized topologies ($\alpha\alpha\alpha$, $\beta\alpha\beta\beta$,$\alpha\beta\beta\alpha$, $\beta\beta\alpha\beta\beta$) and the naturally occurring villin protein. The stability score is the difference between the measured EC_{50} of the actual protein and its predicted EC_{50}. Here, EC_{50} is the protease concentration at which 50% of cells pass, measured on a $\log_1 0$ scale.

In the case designing stable *de novo* proteins, the design variable $\mathbf{x}$ corresponds to the sequence of amino acids for the protein, and the proxy objective model $f_\theta(\cdot)$ is chosen to be a neural network that takes the sequence as input and outputs a real-valued stability score. This is a broad paradigm that encapsulates many different algorithms.

3 Design Choices In Data-Driven Optimization of Proteins

Applying data-driven offline optimization into protein design requires answering several key design questions:

1. **Evaluation**: How can we assess hyperparameters, model training decisions, and generated sequences in the absence of wet lab evaluation?
2. **Training data generation**: How can we create training datasets that are better suited for model-based optimization, starting from large amounts of non-ML oriented wet lab experimental data?
3. **Decision decisions for training and optimization**: How can we tune the design decisions of our offline data-driven optimization method? What metrics should we track, how should we cross-validate?

In this work, we present a set of guidelines and recommendations to serve as a toolkit for practitioners aiming to apply offline model-based optimization for protein design.

3.1 Curating Assay-Labelled Data for Training

The data distribution intricately controls the effectiveness of data-driven optimization algorithms. For instance, if the training distribution is supported on a narrow manifold in the space of protein sequences, running optimization can relatively easily find off-manifold, invalid protein sequences (Trabucco et al., 2021b). This is often the case in biological settings, where data generated for experimental purposes are derived from mutagenesis experiments of a few wildtype sequences, resulting in highly clustered datasets. To prevent this, we recommend that practitioners ensure that the training dataset has a broad coverage of the search space. This may manifest in how data is generated and encouraging sequence diversity, and resampling the assay-labelled data (e.g. while partitioning the data into the training and validation sets) such that it is as uniform as possible.

Utilize broad training data for training. To obtain a broad training dataset, we first run k-means clustering on the proteins in the assay-labeled dataset. Then, to obtain a broad enough training dataset, we use a simple heuristic: we first visualize the clusters on a t-SNE plot, and then pick the subset that are the most spread out for training. The remaining clusters are used for cross-validation.

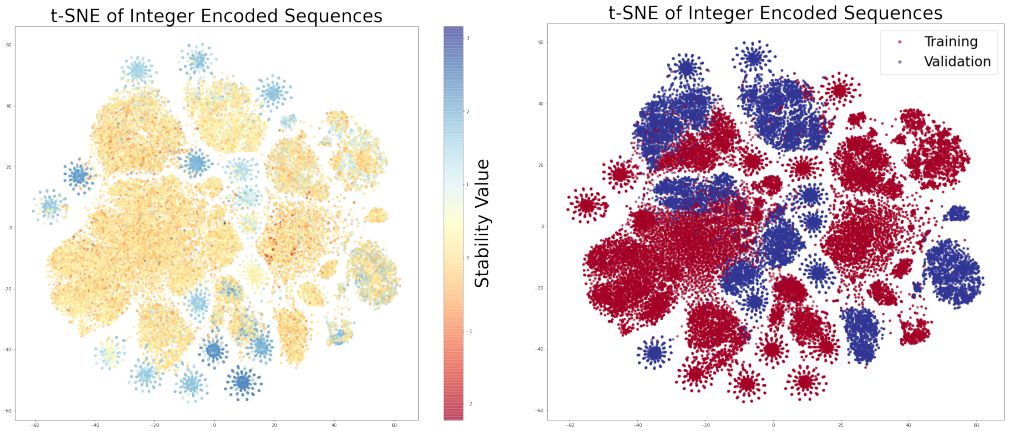

Figure 1: t-SNE plot of dataset, colored by the stability value and colored by the data split

Carefully splitting the data into a training and validation set using this clustering-based approach also ensures that we can reliably assess the model's performance and improved its ability to generalize to proteins derived from a different wildtype sequence. For example, if we randomly split the data into a training and validation set, proteins derived by performing single-site mutations starting from the same wildtype would be likely divided among training and validation subsets, which may provide an easier "short-cut" for trained models which fail to capture intricate sequence features, thus failing to give rise to better optimization performance.

3.2 Ensuring Reliability of Evaluation Via Oracles

As discussed in Section 2, we utilize computational oracles to validate the efficacy of our data-driven optimization procedure. The computational oracles we consider are neural network models learned from data, and therefore are not guaranteed to make accurate predictions on every protein. We will query these oracles on protein designs found after optimization, that are not seen in the training dataset. Therefore, it is crucial to ensure that the design model does not end up "exploiting" the inaccuracies in the oracle, or else our oracle-based evaluation procedure will prefer novel proteins that appear to perform well under the (inaccurate) oracle, but are actually quite unstable. In this section, we will describe some recommendations and protocols for training computational oracles, that minimize the possibility of such oracle exploitation.

Train multiple oracles and track their agreement. To minimize the risk of oracle exploitation, we suggest that a practitioner trains not one, but an ensemble of multiple computational oracles. Additionally, we recommend that the oracles in this ensemble differ in their functional forms, i.e., they utilize different activation functions (e.g., ReLU vs tanh), different neural network layers (e.g., linear layers, self-attention layers, convolutional layers) and vary in model sizes. Such functionally different oracles will differ in their extrapolation behavior, and hence are likely to not be exploited simultaneously. Therefore, we recommend evaluating the performance of various MBO techniques in terms of an aggregate prediction of this oracle ensemble along with the degree of their agreement. We recommend disregarding designs on which oracles disagree even if the aggregate oracle-predicted score is high.

Train oracles on a larger dataset, with appropriate cross-validation. We can further reduce the risk of exploitation by making sure that the oracles are at least as accurate as the models we will use for optimization. To achieve this, following the strategy from Trabucco et al. (2021a), we recommend that each oracle is trained on a dataset larger than the one used for training the design model. Additionally, we wish to track overfitting and underfitting when training the oracles, and attempt to modify the network capacity and add regularization to improve the oracle's validation performance. We outline the oracles we use in Table 1. Note that several of them consist of dropout layers to prevent overfitting and each of them is early-stopped during training at the point of lowest validation error.

Train some oracles exclusively on data which is not used for optimization. To further minimize the risk of exploitation of the oracle models, we also recommend training at least a subset of the oracles on data that is not utilized for training the surrogate in model-based optimization.

3.3 Workflows and Design Choices for Offline Optimization Algorithms

As discussed in Section 2, we focus on the family of offline MBO algorithms that first train a model of the objective function $f_\theta(\mathbf{x})$, that we call a design model, and then optimize it with respect to the input protein $\mathbf{x}$. Learning such a model and optimizing it presents several important design decisions, including but not limited to, the choice of the optimizer, workflows to tune model capacity, and regularization. We present our recommendations surrounding these decisions in the section below.

Discrete optimizer for optimizing discrete protein sequences. Since the space of proteins is discrete, we recommend using a discrete optimizer for obtaining optimized protein designs. In our experiments, we utilize a very simplistic formulation of the discrete

Architecture	Trained On	No. of Parameters	Validation Loss	Spearman
Fully-Connected	All Data	7216053	0.17	0.881
Fully-Connected	Validation Data	7216053	0.19	0.757
Transformer	All Data	668781	0.19	0.666
Transformer	Validation Data	668781	0.19	0.698
LSTM	All Data	2137581	0.16	0.813
LSTM	Validation Data	2137581	0.18	0.755
CNN	All Data	3474537	0.18	0.718
CNN	Validation Data	3474537	0.18	0.747

Table 1: Details of oracle models used for evaluation. The first three columns detail the model architecture, the data that the model was trained and validated on, and the size of the model. "Validation data" refers to the data from the split described in Section 3.1 and depicted in Figure 1). For each model, we report the validation loss of the model from an 80/20 split of the data in the "Trained On" column. Finally, we also report the Spearman correlation of the model on the validation data set, again where "validation data" refers to the data from the split described in Section 3.1 and depicted in Figure 1.

optimizer that optimizes a protein in several rounds starting from a given protein $\mathbf{x}_0$: **(1)** randomly perturb different positions in $\mathbf{x}_0$ with a specified mutation rate, **(2)** evaluate the resulting protein sequences $\tilde{\mathcal{X}}_0 := \{\tilde{\mathbf{x}}_0^1, \tilde{\mathbf{x}}_0^2, \cdots, \}$ under the learned model $f_\theta(\cdot)$, **(3)** retain the top-k of the modified sequences $\tilde{\mathcal{X}}_0$ based on rankings under f_θ, and **(4)** finally repeats the process. This resembles the directed evolution paradigm which has been traditionally successful in protein engineering. In contrast to optimization in some continuous embedding space or via gradient descent like in prior work (Trabucco et al., 2021b), this sequential discrete optimization procedure enables us to impose additional locality constraints during optimization, such as constraining the number of mutations we wish to perform, which can be crucial depending upon the experimental setting of interest.

Cross-validation metrics for tuning overfitting and underfitting. To attain good optimization performance, we must precisely tune the capacity and architecture of the design model $f_\theta(\mathbf{x})$. In addition to the standard approach of using validation error on a held-out dataset to determine overfitting and underfitting, we also recommend that the practitioner track other metrics which are more reflective of optimization performance. For instance, since optimization performance depends on relative values of different proteins, we recommend measuring how closely the ranking of various proteins under the design model matches the ground-truth stability values. In our experiments, we attain this by measuring the Spearman rank correlation between the design model and the ground truth objective on a held-out validation set for ranking.

Add conservative regularization, if needed. While our guidelines prescribe training the design model $f_\theta(\mathbf{x})$ on a broad training dataset, the design model may still suffer from generalization failures common to supervised regression models. To computationally explore new regions of the sequence space, it is intuitive to move further away from the data distributions already experimentally explored to create more sequence diversity, but will run into greater risk of model failure. Optimizing the protein against the output of such a prediction model may still produce proteins that are invalid and unstable, especially when trying to optimize proteins starting from an already good protein. To alleviate this issue, we suggest learning a *conservative* model of the objective function, that is trained via a regularized supervised regression procedure following the COMs method of Trabucco et al. (2021b). This conservative regularizer (Equation 1) penalizes the value of the design model on unseen and potentially invalid proteins $\mu(\mathbf{x})$ that appear promising under the learned model $f_\theta(\cdot)$, preventing the discrete optimizer from designing proteins which appear promising under the learned design model, but do not actually perform well.

$$\min_\theta \quad \underbrace{\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\left[(f_\theta(\mathbf{x}) - y)^2\right]}_{:= \text{ supervised loss}} + \alpha \underbrace{\left(\mathbb{E}_{\mathbf{x}\sim\mu}\left[f_\theta(\mathbf{x})\right] - \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\left[f_\theta(\mathbf{x})\right]\right)}_{:= \text{ conservative regularizer}}. \tag{1}$$

While explicit conservative regularization may not be needed when optimizing a protein sequence sampled uniformly at random from the training dataset only upto a few mutations, we find in our experiments that an adequate amount of conservatism is needed if we wish to attempt to improve the high-scoring protein sequences observed in the data.

4 Evaluation and Analysis

In this section, we present some concrete instances of using guidelines and metrics proposed in Section 3 in making effective design choices for obtaining stable proteins under a variety of testing conditions that would commonly arise in realistic scenarios.

4.1 Test Scenario 1: Improving the Stability of a Randomly Sampled Protein

The goal of our first task is to improve the stability of a randomly sampled protein sequence drawn from the distribution of the dataset. Such a scenario arises in several settings where a designer only needs to locally improve a sequence, or when the goal is to create a diverse set of sequences which pass a certain threshold of stability for downstream refinement for a more specific property. We randomly sample 300 sequences from the held-out validation dataset and evaluate various approaches in terms of the average improvement in stability score and the proportion of sequences that are improved, as measured by the oracles, on an average. In this task, we found that the oracles described in Table 1 were in agreement on the optimized protein designs, indicating that the oracles are likely not exploited.

Model	Average Improvement	Success Rate
Transformer	**0.201**	**0.683**
Transformer w/ conservative	0.061	0.589
Small LSTM	0.072	0.604
Small LSTM w/ conservative	0.011	0.521
Big LSTM	0.164	**0.693**
Big LSTM w/ conservative	0.049	0.570
Delta LSTM	0.102	0.596
Delta LSTM w/ conservative	0.097	0.583

Table 2: Comparison of the capacity and architecture of the design model and the design choice to add conservative regularization on test scenario 1: improving random sequences.

In terms of algorithmic design choices, we compare design models with different network architectures and trained with or without conservative regularization. For all design models, we generate the final design sequences by running 12 iterations of discrete optimization with a mutation rate 0.02 and a pool size of 30. For these sequences, we evaluate the stability scores using our oracles and report two metrics: average improvement and success rate. The average improvement measures the average difference of the optimized sequences compared to the corresponding starting sequence. The success rate captures the ratio of the optimized sequences achieving a higher stability score than their starting sequences. We present the results in Table 4.1. Overall, for all variations, we are able to obtain a positive average improvement. We first considered the more standard LSTM ("Small LSTM" and "Big LSTM" in Table 4.1) and Transformer ("Transformer" in Table 4.1)) architectures for representing the design model. We found that a naïve design model, trained without conservative regularization performed the best in terms of success rate and average improvement.

For the Transformer and both the LSTM architectures, we found that applying any form of conservative regularization resulted in worse performance, both in terms of the final optimization performance, as well as in terms of worse train and validation rank correlation. This indicated that conservative regularization in this scenario leads to poor optimization and generalization performance and the choice of model architecture when applying conservative regularization is a critical choice. To alleviate the poor generalization behavior due to conservatism, we utilized a model that specifically models the improvement in stability obtained as a function of the mutations made to the protein sequence, which we refer to

as the "Delta" model in Table 4.1. This model is better with conservative regularization, but is still somewhat worse than not applying conservatism. This evidence indicates that either conservatism is not required in this setting, when the goal is to improve random sequences sampled from the training data distribution or naïve sequence models such as Transformers and LSTMs may not be the best choice of architecture when using conservative regularization. A study to answer this question is an interesting avenue for future work.

4.2 Test Scenario 2: Improve Stability of the Best Protein Sequences

In this task, we start from the 300 most stable proteins from the holdout set as the starting point and attempt to improve their stability even further. This setting is practically applicable when maximizing stability is more desirable than generating a diverse set of stable protein candidates that pass a certain threshold. For example, a protein designer may be interested in the design of a protein which retains its stability under extreme conditions such as high enzymatic concentrations or temperatures, and thus achieving maximal stability may be a more desirable aim. This task also reflects the use cases of offline optimization for other properties aside from stability, where simply optimizing for the highest-possible value is the primary aim.

We run the proposed discrete optimization procedure to produce the final design sequences and evaluate them according to the average improvement and success rate metrics as discussed previously in Test Scenario 1. We present our results in Table 4.2. Overall, we see that we are unable to produce significant improvement over the best protein in this task. We hypothesize that this is because the top performing sequences in the dataset are likely already very stable and therefore it is hard to improve beyond them.

We would expect that conservative regularization, that constrains the optimization to be close to the sequences observed in the data should perform better in this task, by performing no mutations to the top sequences at all. However, similarly to what we observed in Testing Scenario 1, conservative regularization with the Transformer and LSTM models led to worse validation Spearman correlation on the held-out set and a worse improvement in terms of stability performance under the learned oracles. By utilizing the more expressive Delta model, we find that the magnitude of the decrease in stability from the top sequences were minimal, and this model chose to take the minimum number of mutations compared to the Transformer and LSTMs, hinting at the potential of such an architecture to be better at identifying when a significant improvement may not be possible.

Model	Average Improvement	Success Rate
Transformer	**0.001**	**0.551**
Transformer with COMs	-0.104	0.419
Small LSTM	-0.127	0.348
Small LSTM with COMs	-0.130	0.316
Big LSTM	**0.010**	**0.523**
Big LSTM with COMs	-0.101	0.362
Delta LSTM	-0.062	0.431
Delta LSTM with COMs	-0.003	0.517

Table 3: Comparison of the capacity and architecture of the design model and the design choice to add conservative regularization on test scenario 1: improving top sequences.

4.3 Test Scenario 3: Comparing Against Saturation Mutagenesis Experiments

The final round of the dataset curated in Rocklin et al. (2017) starts with a final subset of favorable designs, and for each design, perturbs every single position to all other possible amino acids to create $19 \times L_{protein}$ possibilities. These site-saturation mutagenesis (SSM) protocols are often used in protein engineering for a more conservative tinkering refinement. We aim to see if our method can assist with a more successful "final tinkering" of sequences than this exhaustive saturation mutagenesis approach. Specifically, we hold out 6 parent

sequences and their full SSM sequences from the design model, and we run optimization on our design models starting from these 6 sequences. We measure the same success rate of produced sequences which are more stable than the parent sequence by oracle evaluation, and compare against the experimentally measured values from the SSM round. The optimization process is kept the same as in test scenarios 1 and 2. Results are shown in Table 4.3. Overall, the MBO approach achieves better performance as compared to the experimental SSM round, though there is no distinct winner with respect to the choice of architecture and conservatism. This task may be of utility for integration with current mutagenesis protocols, as it provides some guidance for multi-site mutagenesis and can be made to more closely emulate saturation mutagenesis by adjusting the mutation rate.

Model	1	2	3	4	5	6
SSM (data)	0.284	0.495	0.369	0.241	0.344	0.139
Transformer	0.350	0.568	0.120	0.178	0.292	**0.676**
Transformer with COMs	0.050	0.422	0.140	**0.308**	**0.422**	0.088
Small LSTM	0.244	0.140	0.022	0.018	0.022	0.328
Small LSTM with COMs	0.262	0.278	0.042	0.028	0.042	0.264
Big LSTM	0.435	0.500	0.215	0.180	0.235	**0.665**
Big LSTM with COMs	**0.510**	**0.745**	0.260	0.225	0.315	0.310
Delta LSTM	0.308	0.410	0.138	0.074	0.136	0.634
Delta LSTM with COMs	0.368	0.550	**0.416**	0.142	0.312	0.498

Table 4: Comparing the success rate of sequences optimized using the design model against sequences obtained via SSM, starting from 6 parent sequences in the SSM round of the dataset. The 6 columns correspond to the 6 parent sequences in the SSM round of the dataset: `EEHEE_rd3_0037`, `HHH_rd2_0134`, `HHH_rd3_0138`, `EHEE_0882`, `EHEE_rd2_0005`, `villin`. The naming prefix indicates the structure of the protein, where `E` corresponds to β-sheets, and `H` corresponds to α-helices. Villin is a naturally occuring miniprotein. The SSM row represent the success rate of the experimental SSM round in the dataset; overall, using model-based optimization achieves better performance.

5 Conclusion

In this work, we present a systematic study of offline data-driven optimization for protein design. We introduce a general framework for offline data-driven optimization methods and identify various of design choices under this framework such as model capacity and regularization. Along with these design choices, we also consider several important factors for the workflow of applying offline data-drive optimization algorithms, especially in the absence of wet lab validation. We analyze these design choices and workflow factors on the task of *de novo* stable protein design, and validate them under a range of test scenarios. We hope that our results can serve as an guide for biology practitioners to better adopt data-driven offline optimization methods in the field of protein engineering.

References

Minkyung Baek, Frank DiMaio, Ivan Anishchenko, Justas Dauparas, Sergey Ovchinnikov, Gyu Rie Lee, Jue Wang, Qian Cong, Lisa N Kinch, R Dustin Schaeffer, et al. Accurate prediction of protein structures and interactions using a three-track neural network. *Science*, 373(6557):871–876, 2021.

David Brookes, Hahnbeom Park, and Jennifer Listgarten. Conditioning by adaptive sampling for robust design. In *International conference on machine learning*, pages 773–782. PMLR, 2019.

Drew H Bryant, Ali Bashir, Sam Sinai, Nina K Jain, Pierce J Ogden, Patrick F Riley, George M Church, Lucy J Colwell, and Eric D Kelsic. Deep diversification of an aav capsid protein by machine learning. *Nature Biotechnology*, 39(6):691–696, 2021.

Justin Fu and Sergey Levine. Offline model-based optimization via normalized maximum likelihood estimation. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL `https://openreview.net/forum?id=FmMKSO4e8JK`.

Brian L Hie and Kevin K Yang. Adaptive machine learning for protein engineering. *Current opinion in structural biology*, 72:145–152, 2022.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873): 583–589, 2021.

Aviral Kumar and Sergey Levine. Model inversion networks for model-based optimization. *NeurIPS*, 2019.

Gabriel J Rocklin, Tamuka M Chidyausiku, Inna Goreshnik, Alex Ford, Scott Houliston, Alexander Lemak, Lauren Carter, Rashmi Ravichandran, Vikram K Mulligan, Aaron Chevalier, et al. Global analysis of protein folding using massively parallel design, synthesis, and testing. *Science*, 357(6347):168–175, 2017.

Sam Sinai and Eric D Kelsic. A primer on model-guided exploration of fitness landscapes for biological sequence design. *arXiv preprint arXiv:2010.10614*, 2020.

Sam Sinai, Richard Wang, Alexander Whatley, Stewart Slocum, Elina Locane, and Eric D Kelsic. Adalead: A simple and robust adaptive greedy search algorithm for sequence design. *arXiv preprint arXiv:2010.02141*, 2020.

Brandon Trabucco, Xinyang Geng, Aviral Kumar, and Sergey Levine. Design-bench: Benchmarks for data-driven offline model-based optimization, 2021a. URL `https://github.com/brandontrabucco/design-bench`.

Brandon Trabucco, Aviral Kumar, Xinyang Geng, and Sergey Levine. Conservative objective models for effective offline model-based optimization. In *International Conference on Machine Learning*, pages 10358–10368. PMLR, 2021b.