STraP: Self-Training for Proteins

Arbaaz Muslim
Nilah Ioannidis

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-110
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-110.html

May 11, 2023

Acknowledgement

Thank you to Professor Nilah Ioannidis for giving me the opportunity to pursue my deep interest in machine learning in biology. Her invaluable insights and support throughout were crucial to this project.
Thank you to eyes robson for being as generous with their time as they were with their knowledge.
Thank you to my family and friends, who made this more of a group effort than the title page makes it seem.

STraP: Self-Training for Proteins

Arbaaz Muslim

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee

Nilah Ioannidis

5/11/23

(Date)

⋆ ⋆ ⋆ ⋆ ⋆ ⋆ ⋆

Laura Waller

5/7/23

(Date)

STraP: Self-Training for Proteins

by

Arbaaz Muslim

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Spring 2023

STraP: Self-Training for Proteins

Abstract

STraP: Self-Training for Proteins

by

Arbaaz Muslim

Master of Science in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Nilah Ioannidis, Chair

Protein engineering is a field with the potential for immense impact in a broad range of fields such as agriculture, medicine, and manufacturing. However, manually searching for proteins (or equivalently, amino acid sequences) with desirable properties by generating and testing large numbers of candidate sequences in experimental assays is incredibly resource-intensive. Computational approaches to model protein fitness, particularly few-shot learning approaches that can leverage a limited quantity of experimental assay-labeled data for training, are therefore highly desirable. In this work, we explore a computational approach that combines prior work in protein language modeling using large language models with the few-shot learning technique of self-training, which iteratively generates pseudo-labels for unlabelled sequences during fine-tuning to enhance the accuracy of a model's predictions despite sparsely available labeled data. Here, we perform initial tests of self-training for proteins and propose follow-up studies to further explore this approach.

To my family

Contents

List of Figures

List of Tables

Acknowledgments

Thank you to Professor Nilah Ioannidis for giving me the opportunity to pursue my deep interest in machine learning in biology. Her invaluable insights and support throughout were crucial to this project.

Thank you to eyes robson for being as generous with their time as they were with their knowledge.

Thank you to my family and friends, who made this more of a group effort than the title page makes it seem.

Chapter 1

Introduction

We owe our everyday lives to proteins - from catalysts to antibodies, proteins are crucial to sustaining human life. The ability to efficiently engineer proteins tailored to particular use cases would have a large impact in fields such as medicine, agriculture, and manufacturing (among others) [1]. The "directed evolution" approach to protein engineering relies on generating a large number of candidate amino acid sequences and then repeatedly performing experimental assays on these sequences [1]. Such assays quantify a candidate protein's effectiveness at accomplishing a target task. The general term for this quantity is "fitness". In effect, assays can be thought of as algorithms that take in an amino acid as input and provide fitness as an output.

A critical aspect of protein engineering is navigating the trade-off between an assay's fidelity (how accurately the resulting fitness value represents the target task in the real world) with its throughput (how many proteins can undergo the assay at one time). The two factors are inversely related - high fidelity assays typically require more resources, and therefore have low throughput, and vice versa. As a result of this inverse relationship, assays are performed iteratively. An initial, large number of candidate proteins undergo relatively simple assays that are easier to carry out. Those with acceptable fitness values then move onto the next round, where more resource-intensive, higher fidelity assays are performed [1].

However, cost is a major limiting factor for this approach. Developing and performing assays is an inherently expensive procedure in terms of time, effort, and money. Another shortcoming is the gap between a protein's perceived fitness in the lab and its actual performance when applied to its real-world target task. Due to the nature of assays being controlled processes that occur in a lab, a protein's fitness is a proxy for its real-world effectiveness. The process of protein engineering aims to optimize fitness as measured by the available assay, which often times does not accurately reflect its performance on the actual target task [1]. In addition, many

classes of proteins have no associated assays [1].

We mentioned earlier that assays can be thought of as algorithms that take in an amino acid sequence as input and produce fitness as an output. Thus, in a computational approach to protein engineering, directed evolution in the lab could be augmented (and potentially replaced) by a machine learning algorithm that predicts fitness labels for inputted amino acid sequences. At first glance, it would seem that this approach is subject to the same constraints as with directed evolution - machine learning models require a set of training data, which in this case would have to come from assays performed in the lab. Therefore, we need a machine learning approach such as few-shot learning, which is a form of machine learning that relies on limited training data. It would also be ideal to leverage unlabelled sequences to enhance performance by using semi-supervised learning, an approach that involves a relatively small quantity of labelled data and a relatively large quantity of unlabelled data in model training.

Thus, our desired machine learning approach would be few-shot, so that it can leverage a small number of labelled sequences, and simultaneously semi-supervised, so that it can leverage a large number of unlabelled sequences. Self-training [10] is a technique that fulfills both requirements and has been applied to a variety of natural language datasets. Notably, [10] shows that applying self-training to the SNLI dataset with only 8 labeled examples produces an accuracy improvement of 21.3% using the standard BERT model and an improvement of 26.2% using $BERT_{LARGE}$. The authors of [10] summarize self-training as the following:

> *...at each iteration, the base model is fine-tuned using the available labeled data for the target task. Then, the resulting model's predictions on unlabeled examples are used as pseudo-labels to augment the original labeled data set. The newly formed labeled data set is then used to learn a better model in the next iteration, and this procedure is repeated for a number of iterations until a stopping criterion is reached.*

In order to translate this natural language technique to a protein engineering context, we make a number of modifications. In contrast to the BERT models used in [10], we use Evolutionary Scale Modeling transformers (for the specific models, see **Methods**) provided by Facebook AI Research. These are large language models like BERT that have been pre-trained on protein sequence data and accept amino acid sequences as input. In addition, in contrast to the natural language datasets used in [10], we use a subset of the protein fitness datasets from [4]. These datasets span a variety of biological contexts, such as transcriptional activity, peptide binding, and ampicillin resistance, among others. Each dataset includes the fitness value of

the wild-type protein as well as fitness values for a number of mutant sequences with single site substitutions. The UBE4B dataset includes mutant sequences with multiple mutations as well. In addition, each dataset includes a multiple sequence alignment (MSA) featuring sequences that share evolutionary history with the wild-type sequence. The MSA for each dataset was used as our source of unlabelled data when applying self-training. For more details on the datasets, see **Methods**.

Chapter 2

Methods

2.1 Data

Datasets

We use a subset of the data from [4]. There were two main considerations in determining which datasets to use - ensuring that a variety of biological contexts were represented in our subset and ensuring that performing self-training on the chosen data conformed with time and computational constraints.

For every dataset, we ran experiments using 64, 128, 256, and 512 labelled sequences. This meant that the dominant factor that determined training time for self-training was the number of unlabelled sequences. In order to ensure computa-

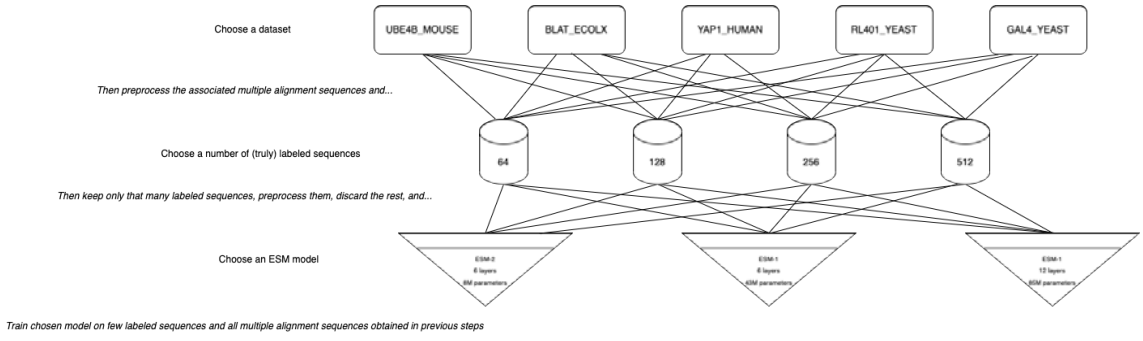

Figure 2.1: Overview of experimental design. Each path through the decision tree denotes one experiment.

tional feasibility, we chose the 5 datasets with the smallest numbers of unlabelled sequences as described below.

Protein	Measurement	UniProt ID	Total # of labeled sequences	Total # of MSA sequences	Reference
UBE4B (U-box domain)	Ligase activity	UBE4B_MOUSE	32290	6533	Starita et. al., PNAS, 2013
β-lactamase	Ampicillin resistance	BLAT_ECOLX	4807	7701	Stiffler et. al., Cell, 2015
YAP1 (WW domain 1)	Peptide binding	YAP1_HUMAN	319	8707	Araya et. al., PNAS, 2012
Ubiquitin	Growth	RL401_YEAST	1161	10145	Roscoe et. al., JMB, 2013
GAl4 (DNA-binding domain)	Transcriptional activity	GAL4_YEAST	1123	11681	Kitzmann et. al., Nature Methods, 2015

Table 2.1: Chosen datasets for experiments

Sourcing unlabelled sequences

The use of unlabelled sequences is the definitive distinguishing factor between standard fine-tuning and self-training. Based off of [10], we know that self-training works best when the number of unlabelled sequences is significantly larger than the number of labelled sequences ("$M << N$" [10]) and when the unlabelled sequences come from the same distribution as the labelled sequences.

We had three options for potential sources of unlabelled sequences. The first option was to discard the labels for the labeled sequences that were not included in training. We did not proceed with this method as the amount of unlabelled sequences it produced was often only slightly larger than the labelled sequences we trained on. The second option was to generate sequences by artificially inserting mutations into the wild-type amino acid sequence. While this method did not usually suffer from the issue of lacking magnitude, it was not guaranteed to produce sequences corresponding to naturally occurring proteins. This meant that it was not guaranteed to produce sequences that were "in distribution" compared to our labelled datasets. The final option was to extract sequences from the provided multiple sequence alignment (MSA) for each protein. This option consistently satisfied the magnitude constraint. In addition, the data we obtained from [4] filtered its labeled sequences to "exclude sequences with mutations at positions that have more than 30% gaps in MSAs to focus on regions with sufficient evolutionary data" [4]. We interpreted the inclusion of evolutionary data as a best-effort attempt at satisfying the distribution constraint. For these reasons, we used MSA sequences as the sources for the unlabelled sequences in our experiments.

Some pre-processing was applied to the MSA sequences before they were used in self-training. Spaces were removed and the ensuing duplicate sequences were dropped. These pre-processing steps led to values in Table 2.1 that differed from the MSA sizes reported in [4].

Defining "few shot"

In order to determine a viable definition of few shot for this work, we needed labelled datasets that were limited in size yet large enough for the resulting validation datasets to be able to convey informative Spearman rank correlation values for the purposes of early stopping.

We drew on [10]'s labelled dataset sizes in order to determine appropriate dataset sizes for our own work. [10] used 8 labelled samples for the data regime that its authors explicitly labelled as "few shot." It also reported results on a logarithmic scale from 8 to 512 in a separate section. We elected to use a logarithmic scale as well, hoping to illustrate the effects of self-training over a wide variety of labelled dataset sizes. However, the nature of Spearman rank correlation motivated us to begin our logarithmic scale at 64 rather than 8. Since we use 20% of our labelled dataset as a validation set, choosing 8 labelled samples would have resulted in a validation observation rather than a validation dataset. A ranking metric would provide no useful information with this setup. We included experiments with a labelled dataset size of 64 more for illustrative purposes than analytical ones, as the validation dataset in that scenario only spans 12 observations. This is also a comparatively trivial size over which to compute Spearman rank correlation.

Ultimately, we were able to determine dataset sizes that were limited in size and capable of providing information-rich Spearman rank correlation values, thereby faithfully recreating a few-shot context that provided a relatively high resolution view of model performance.

Data splits

The way in which we split our data was another point of departure from [10]. Since the experimental setup of both this work and [10] uses early stopping to determine the number of self-training iterations and the number of fine-tuning iterations, both require a set of data for validation purposes. In [10]'s case, this requirement was fulfilled through a labelled development set that was separate from the training set. In this work, we set aside a portion of the already limited training set to serve as a validation set. We chose this setup in order to more accurately reflect the nature of a few-shot learning context, where labelled data is sparse, no matter its intended purpose.

For each dataset, we used 20% of the total labelled sequences available as the test set. We selected a small number of samples (either 64, 128, 256, or 512) from the remaining 80% of labelled sequences and then performed an 80/20 train/val split on this small number.

2.2 Self Training Algorithm

In order to implement our self-training algorithm, we follow the general algorithm provided in [10].

initialization

$t = 0$

Form a base model $\boldsymbol{f}_0$, which is initialized with pre-trained parameters from a pre-training/intermediate fine-tuning stage, and then learn a teacher model $\boldsymbol{f}_1$ by training $\boldsymbol{f}_0$ on the original labeled data set $\boldsymbol{\mathcal{L}}$.

repeat

$t = t + 1$

1. Use the current teacher model $\boldsymbol{f}_t$ to annotate (for $t = 1$) or re-annotate (for $t > 1$) all of the examples in $\boldsymbol{\mathcal{U}}$ to obtain a set $\widetilde{\boldsymbol{\mathcal{U}}}$ of pseudo-labeled examples.

2. Add the whole set $\widetilde{\boldsymbol{\mathcal{U}}}$ of pseudo-labeled examples to the original labeled data set $\boldsymbol{\mathcal{L}}$ to form a new labeled data set.

3. Learn a student model $\boldsymbol{f}_{t+1}$ by training the base model $\boldsymbol{f}_0$ on the current labeled data set and optionally fine-tune it on $\boldsymbol{\mathcal{L}}$. The resulting student model $\boldsymbol{f}_{t+1}$ is used as a teacher for the next iteration.

until *convergence or the maximum number of iterations is reached*

The self-training algorithm from [10]

One major difference in the experimental setup between [10] and this work was the target task, which affected the choice of evaluation metric. [10] tackled a classification task, making accuracy the target metric. This allowed model training with a cross-entropy loss function. In contrast, this work's target metric was Spearman rank correlation. This metric's reliance on rank clearly makes it a non-differentiable function; therefore, we draw upon [4] and adapt its method to "fine-tune the entire Transformer model with fitness labels as done by Rives et al., using the [pseudo log

Model ID	Number of parameters	Number of layers	Pre-training dataset
esm2_t6_8M_UR50D	8M	6	UniRef50/D 2021_04
esm1_t6_43M_UR50S	43M	6	UniRef50/S 2018_03
esm1_t12_85M_UR50S	85M	12	UniRef50/S 2018_03

Table 2.2: Information on ESM models used in this work [7].

likelihood] difference between the mutant and the wild-type as a predictor for fitness" [4, 9].

We use at most 20 self-training iterations maximum in contrast to the 100 used in [10] and 5 maximum fine-tuning iterations in contrast to the 20 used in [4]. These decisions were motivated by computational constraints, but we believe they are justified due to the use of significantly smaller models.

In addition, we use early stopping for both self-training and fine-tuning, relying on the Spearman rank correlation on the validation dataset to determine the number of actual iterations. The importance of doing so is underscored by [4]: "Using Spearman correlation as opposed to validation loss for early stopping is crucial for performance, especially on small data sizes."

2.3 ESM Models

The choice of base model was repeatedly emphasized as a key causal factor for the impressive performance gains provided by self-training in [10]: "an important ingredient in self-training algorithms is the base model f_0 (author's note: see self-training algorithm in 2.2). Successful self-training typically requires a good base model, which can provide a large proportion of "correct" predictions or pseudolabels on unlabeled examples; otherwise, errors can be propagated or magnified in later stages of self-training."

Using the number of parameters in models as a proxy quantity to represent model strength, we experimented with models of three different strengths in order to examine this claim in the context of protein language modeling. All of these models were Evolutionary Scale Modeling transformer models provided by Facebook AI Research [7].

Time and compute constraints were also considerations in the choice of models. These constraints prevented us from using larger ESM models that were closer to the size of the BERT models used in [10], such as the ESM-1b model that was used in [4]. This leaves the door open for a fascinating avenue of further exploration.

2.4 Hyperparameter selection

The two main hyperparameters that needed to be determined for training were learning rate and batch size. Values for both were determined using automatic Pytorch Lightning tuning, using the Spearman rank correlation of the validation dataset as a target metric. In order to address the previously discussed issue posed by using a ranking metric in a few-shot learning context, hyperparameter values were found for each combination of model and dataset using a labelled dataset size (M) of 512 (the maximum labelled dataset size used for experiments). This prevented the issue of incorrectly choosing hyperparameter values based on performance metrics derived from trivially small validation sets. We present the chosen hyperparameter values for each experiment in Table 2.3.

Model ID	Abbreviated Dataset name	Learning Rate	Batch size
esm2_t6_8M_UR50D	UBE4B_MOUSE	2.2908676527677735e-7	410
	BLAT_ECOLX	3e-7	
	YAP1_HUMAN	3e-7	
	RL401_YEAST	7.585775750291837e-8	
	GAL4_YEAST	2.2908676527677725e-5	
esm1_t6_43M_UR50S	UBE4B_MOUSE	3e-7	
	BLAT_ECOLX	3e-7	
	YAP1_HUMAN	3e-7	
	RL401_YEAST	7.585775750291837e-8	
	GAL4_YEAST	7.585775750291837e-8	
esm1_t12_85M_UR50S	UBE4B_MOUSE	2.2908676527677735e-7	
	BLAT_ECOLX	1.5848931924611133e-7	
	YAP1_HUMAN	3e-7	
	RL401_YEAST	1.2022644346174132e-6	
	GAL4_YEAST	9.120108393559096e-8	

Table 2.3: Chosen hyperparameter values for experiments

2.5 Establishing a baseline

We chose to use standard fine-tuning with early stopping as a baseline method. This baseline method differed from self-training in that it only used the set of truly labelled data. We used the same number (5) of maximum fine-tuning iterations for the baseline method and the fine-tuning that occurred within each iteration of self-training.

An earlier experimental design performed fine-tuning by retrieving the embeddings from ESM models and then passing them through a separate neural network that predicted a fitness value. In this setup, both the ESM model and the ensuing neural network were trained (i.e. none of the weights involved were frozen). However, this method required much more time and compute and was ultimately abandoned in favor of the method used in [4].

2.6 Metrics

A variety of metrics were used to analyze the impact of self-training. Of these, Spearman rank correlation was the primary metric for measuring success, as it has been established as a standard mode of evaluation for protein fitness in past works [2, 4].

Spearman Rank Correlation

$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$ where
d_i = the difference between the two ranks of the i$^{\text{th}}$ observation
n = the number of samples

Spearman rank correlation is a common metric used to evaluate models in protein fitness prediction. Given that the aim is to augment or replace the intensive nature of protein assays, it may seem odd that a ranking metric is used instead of some kind of loss function that captures the distance between assay values that are predicted computationally and assay values that are obtained experimentally. However, in most protein engineering contexts, the aim is to determine which sequences among a pool of search candidates are the best at performing a desired task. In other words, the real-world use case that motivates computational protein fitness prediction defines success in relative terms, making a ranking metric a natural fit.

The use of a ranking-based target metric poses a unique challenge in the context of few-shot learning - the smaller a dataset, the less information a ranking metric conveys. It is trivially simple to correctly (or incorrectly) rank a handful of observations. In such scenarios, Spearman rank correlation provides a very low-resolution view of a model's performance. This side effect of choosing Spearman rank correlation as a target metric influenced this work's interpretation of the term "few-shot" and the way in which we performed data splits.

Normalized Discounted Cumulative Gain

For a query, the normalized discounted cumulative gain, or NDCG, at a particular rank p is computed as:

$$NDCG_p = \frac{DCG_p}{IDCG_p}$$

$$DCG_p = \sum_{i=1}^{p} \frac{rank_i}{\log_2(i+1)} = rel_1 + \sum_{i=2}^{p} \frac{rank_i}{\log_2(i+1)}$$

$$IDCG_p = \sum_{i=1}^{p} \frac{rank_i}{\log_2(i+1)}$$

[5]

In contrast to the relatively egalitarian Spearman rank correlation, normalized discounted cumulative gain (NDCG) is a ranking metric that gives higher weight to values that have high true ranks [4]. This means that observations that are correctly ranked highly will result in a larger increase in NDCG than correctly ranked lower observations. The opposite is true for incorrectly ranked observations. In the context of protein engineering, NDCG is a useful metric when the focus is on finding protein sequences specifically resulting in the most enhanced fitness relative to the wild-type sequence. In this scenario, when candidate sequences are ranked by fitness, only the topmost ones are of interest, making NDCG a logical metric for measuring performance. Despite the difference in weighting, NDCG and Spearman rank correlation are both ranking metrics at their core. This means that there is often a high degree of correlation between both metrics when they are used to evaluate the same set of results.

Mean Squared Error

$$MSE = \sum_{i=1}^{N} (y_{pred} - y_{true})^2$$

Since Spearman rank correlation and NDCG are ranking measures that reflect relative accuracy, we also included mean squared error as a means of measuring how well the models were able to predict protein fitness in a more objective sense. While this metric is not traditionally used for protein fitness measurement in machine learning, it could help evaluate the models' effectiveness in situations where researchers would want to determine a single protein's fitness value independent of other candidate sequences. In this context, the model's utility would be measured solely by the difference between the actual assay value and the model's output, much like the quantity that mean squared error produces.

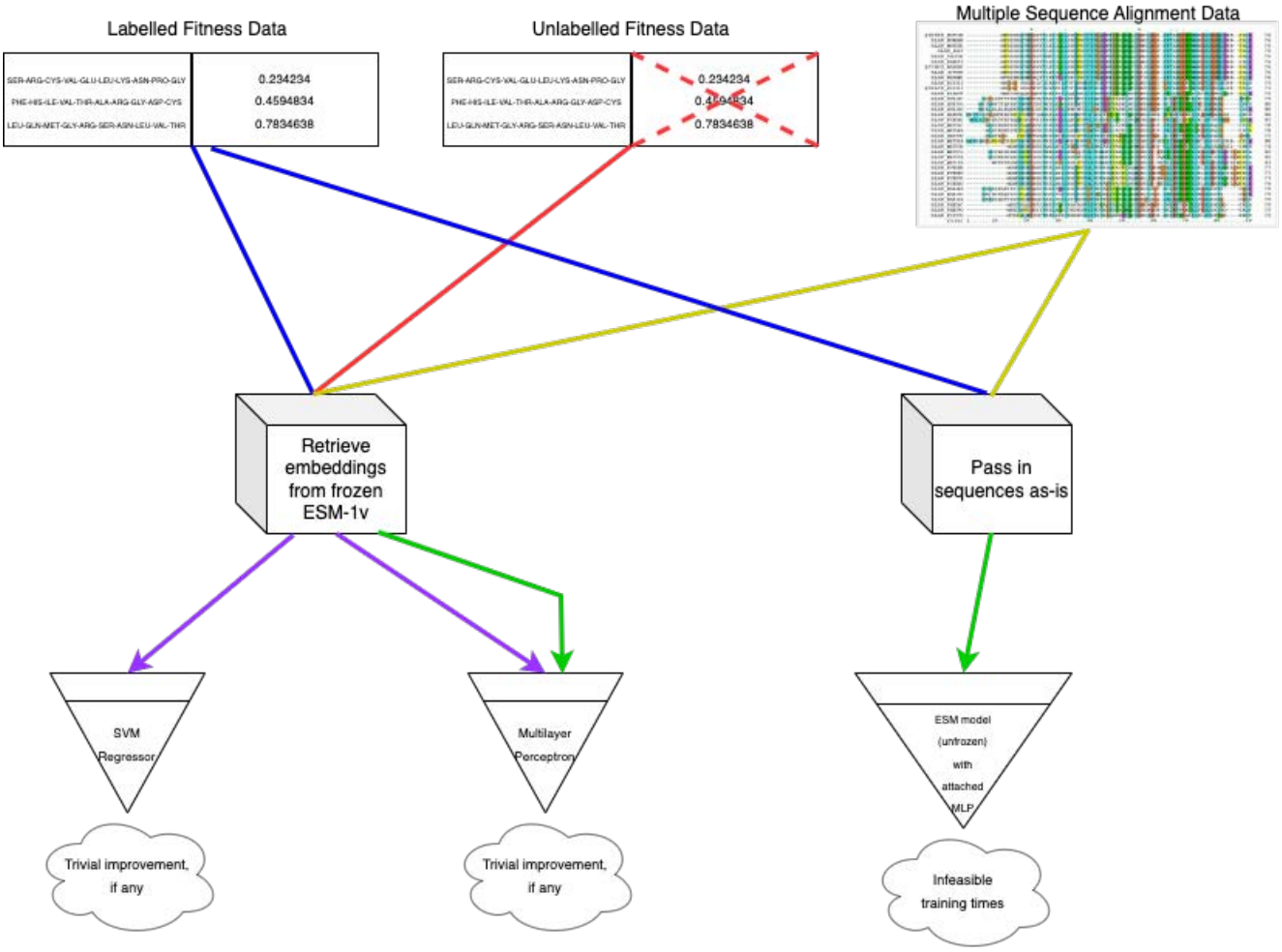

Figure 2.2: Overview of initial implementation attempts. Arrows that are compound colors indicate the data sources that constitute the "new labeled dataset" created by combining the truly labeled data and pseudolabeled data. Arrows of compound colors are ordered in chronological order of implementation from left to right.

2.7 Previous Implementations

The implementation of self-training used in this work went through a number of iterations (`https://github.com/arbaazm1/selftrain-proteins`, `https://github.com/arbaazm1/pl-st-proteins`) before the authors committed to the one described

above.

The very first iteration was an extremely bare-bones, naive implementation of self-training which used embeddings from the pre-trained ESM-1v model as input to an SVM regressor. Importantly, the ESM-1v model's weights remained frozen throughout the implementation - only the SVM was undergoing self-training. This method featured the use of only labelled data (directly from the DeepSeq paper [8]), which was preprocessed by removing duplicate sequences and sequences with NaN labels. Instead of the MSA sequences we ultimately used for unlabelled data, we selected 512 labelled sequences from the set of all preprocessed labelled data and then discarded the labels for the rest, in effect creating a set of unlabelled data. This approach unsurprisingly yielded unimpressive results, and we conjectured that this was because the use of a strong base model was a major contributing factor to the success of self-training in [10]. In comparison, the SVM used in this implementation was nowhere near as complex.

In subsequent iterations, we aimed to tackle the underlying issue of lacking model complexity by using small neural networks (1-2 hidden layers). These yielded similar results. We then decide to make a change in the data being used as we realized [10] mentioned using a set of unlabelled sequences that was significantly larger than the set of labelled sequences ($M << N$). Our label-discarded set of unlabelled sequences did not follow this rule in most cases.

To tackle the issue of lacking unlabelled data, we used MSA sequences from [8] instead of the sequences with discarded labels that we were using earlier. The MSA sequences were pre-processed by removing MSA-specific symbols (., -, and spaces) and dropping duplicates from the resulting set of sequences. Labelled sequences were pre-processed in the same way as before, by removing sequences with NaN labels and duplicated sequences. The results of this approach illustrated that our choice of base model was still sorely lacking - the BERT models used in [10] (where all of the parameters were being fine-tuned) dwarfed the ESM-1v and multilayer perceptron combination we were using (where only the MLP weights were being fine-tuned, with ESM-1v weights remaining frozen).

We decided to tackle this by leaving no weight frozen in the ESM-1v and MLP combination setup. However, this led to infeasible training times, with only a few self-training iterations being accomplished in a matter of days. There was an attempt to alleviate this issue by re-implementing via Pytorch Lightning (`https://github.com/arbaazm1/pl-st-proteins`), but this ultimately yielded unresolvable out-of-memory errors.

This training time infeasibility motivated another literature review to find other methods of fine-tuning ESM models in particular. This literature review was particularly fruitful, as it provided new methods addressing both prior issues - model

size and data availability. We decided to use smaller ESM models than ESM-1v to address the training time issue. Despite the decrease in model size, the chosen models were still markedly more complex than the SVMs and multilayer perceptrons used earlier, [4] yielded a method of fine-tuning ESM models on fitness data as well as the fitness data that was ultimately used.

Chapter 3

Results

3.1 Initial implementation

We begin by illustrating the results of initial attempts at implementing self-training. These results motivated experimental design changes that culminated in the choices described in **Methods**. The figures in this section show results for the BLAT_ECOLX dataset. Since these initial attempts discarded labels from a subset of the labeled dataset to create an unlabeled set, the BLAT_ECOLX dataset is particularly illustrative since it had one of the highest ratios of unlabelled to labelled samples. Despite being initial attempts, the number of self-training iterations and the method of splitting the data into training, validation, and testing sets match the final implementation. The results illustrated in this section are all averaged over 3 random seeds.

The first attempt used embeddings from ESM-1v with frozen weights and fed these embeddings into an SVM regressor (Figure 3.1). This method provided no significant improvement compared to standard fitting and predicting. The Spearman rank correlations in both scenarios (and therefore the plots) are almost identical. This lack of performance gain can be attributed to a number of shortcomings, which are mentioned in **Methods**.

Future attempts involved replacing the SVM regressor with a small multilayer perceptron. These attempts also did not illustrate much promise with self-training.

3.2 Comparing self-training to fine-tuning

We now compare the performance of self-training using the full approach described in **Methods** to the performance of standard fine-tuning for 5 iterations across 3

Figure 3.1: Comparison of Spearman rank correlation with and without self-training using ESM-1v embeddings and an SVM regressor. Results are reported over 3 random seeds.

different metrics - Spearman rank correlation (Figure 3.2), normalized discounted cumulative gain (NDCG) (Figure 3.3), and mean squared error (MSE) (Figure 3.4).

Effects on Spearman rank correlation

Overall, self-training does not appear to provide the impressive performance gains with this experimental setup as it did in [10]. In most cases, self-training nearly matches or underperforms compared to the baseline fine-tuning procedure. When the computational costs of self-training are taken into account (i.e. many multiples of the time and effort involved in standard fine-tuning), the occasional modest gains provided by self-training are even more insignificant.

This may be attributable to a difference in experimental setup between this project and [10] - while [10] used a development set separate from its training set for validation purposes, this work used a subset of the training data for these purposes. The authors of [10] conjecture that real-world low-resource environments in which separate development sets are unattainable may lead to "lower accuracy in each self-training iteration, [with] these errors [being] compound[ed] through later iterations." However, it is worth noting that the real-world few-shot exploration in [10] still provided nontrivial gains, albeit ones that were worse than when a separate development set was used. We posit that the lack of any notable gains in this setup were due to a combination of two factors - the lack of a development set during self-training and lacking magnitudes in the models and datasets used.

Despite the UBE4B_MOUSE_Klevit2013-nscor_log2_ratio being the only dataset that includes higher-order mutants, it does not appear as if this leads to a pronounced difference in performance. As with other datasets, self-training usually underperforms and occasionally slightly outperforms the baseline fine-tuning method. The only difference is decreased performance when comparing the 6-layer ESM-2 model with 8 million parameters to the 6-layer ESM-1 model with 43 million parameters. As with the trend in other datasets, we expected both baseline and self-training performance to increase with increased model complexity, but it appears that going from the 6-layer ESM-2 model with 8 million parameters to the 6-layer ESM-1 model with 43 million parameters did not follow this trend.

The effect of increasing model size on Spearman rank correlation is apparent - for every dataset, increasing model complexity leads to improved performance for both self-training and the baseline fine-tuning method. This is visible in Fig 3.2 going from left to right for every row. The effect is particularly striking for the datasets YAP1_HUMAN_Fields2012-singles-linear and RL401_YEAST_Bolon2013-selection_coefficient, with baseline Spearman rank correlation rising around 0.4 for the former and around

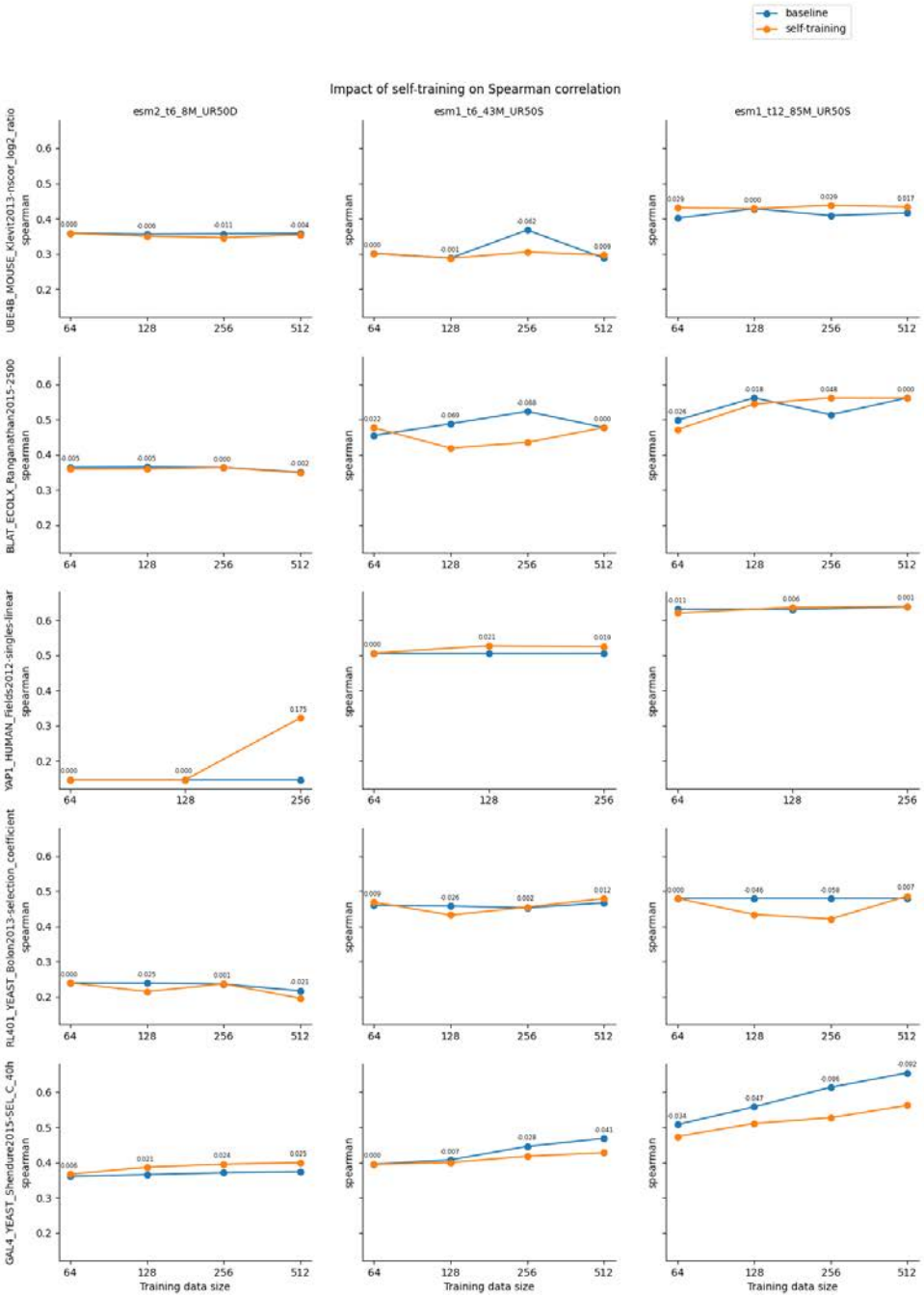

Figure 3.2: Comparison of Spearman rank correlation when applying self-training and standard fine-tuning. Model strength/complexity increases from left to right, and the number of unlabelled sequences increases from top to bottom. On every plot, each point is annotated with the improvement provided by using self-training.

0.2 for the latter when comparing the 6-layer ESM-2 model with 8 million parameters to the 6-layer ESM-1 model with 43 million parameters.

Effects on normalized discounted cumulative gain

The trend for NDCG is similar to that of Spearman rank correlation - performance either matches or underperforms that of standard fine-tuning. Once again, when taking the cost disparity between the methods into account, we conclude that what may appear to be matching performance in terms of the performance metric is actually an underperformance when viewed holistically.

The high correlation between self-training's impacts on Spearman rank correlation and self-training's impacts on NDCG are expected, as both are ranking metrics. However, the two metrics differ in the way that they weigh higher-ranked data points. Spearman rank correlation is a relatively egalitarian measure, with (mis)matches holding equal weight no matter their position in the ranking. In contrast, NDCG places greater weight on higher-ranked data points. Correctly or incorrectly ranking data points with higher true rankings has a greater impact on the resulting metric than when the same is true of data points with lower true rankings [4]. This means that comparing the impact of self-training on Spearman rank correlation with the impact of self-training on NDCG can yield insights on how self-training affects prediction of protein sequences with higher fitness values.

Overall, it seems that self-training is worse at correctly predicting that proteins have enhanced fitness (i.e. correctly predicting highly ranked proteins), as there are multiple instances of self-training NDCG scores underperforming baseline NDCG scores where the opposite was true when viewed through the lens of Spearman rank correlation. This is visible when 512 labelled sequences from UBE4B_MOUSE_Klevit2013-nscor_log2_ratio were used to train the 12-layer ESM-1 model with 85 million parameters, when the 6-layer ESM-1 model with 43 million parameters is trained on BLAT_ECOLX_Ranganathan2015-2500, when the 12-layer ESM-1 model with 85 million parameters is trained on RL401_YEAST_Bolon2013-selection_coefficient, and when the 6-layer ESM-2 model with 8 million parameters is trained on YAP1_HUMAN_Fields2012-singles-linear. The last of these examples is particularly notable, as the NDCG is negative compared to the positive Spearman rank correlation for the same runs.

In addition, the disparity between self-training and standard fine-tuning is more pronounced when viewed through the lens of NDCG. Differences in Spearman rank correlation between the two methods are magnified when NDCG is considered instead.

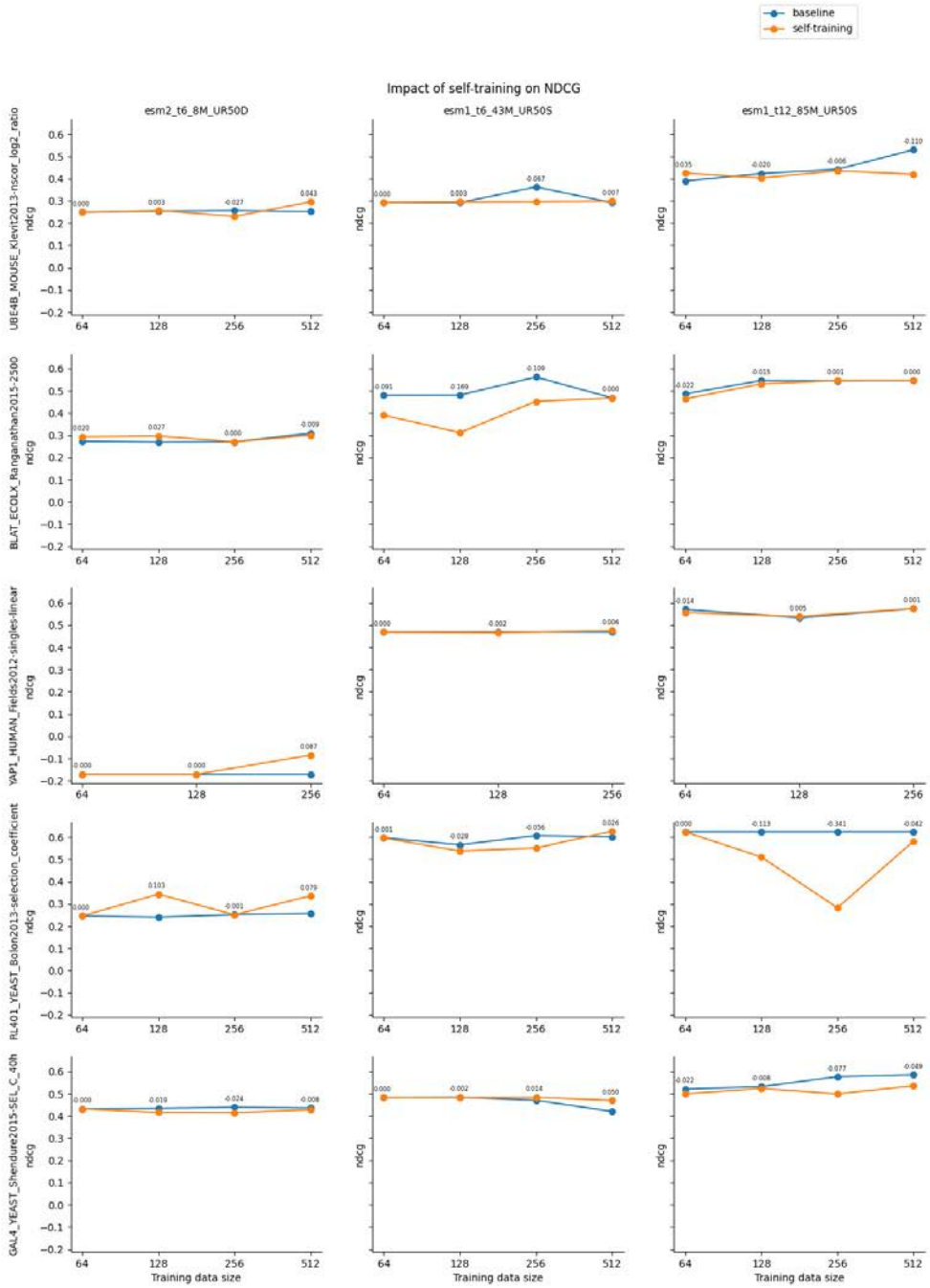

Figure 3.3: Comparison of NDCG when applying self-training and standard fine-tuning. Model strength/complexity increases from left to right, and the number of unlabelled sequences increases from top to bottom. On every plot, each point is annotated with the improvement provided by using self-training.

Effects on mean squared error

In most instances, the mean squared error between both methods was fairly close, with that of self-training being slightly larger. There is relatively high correlation between the Spearman rank correlation and mean squared error, as well as a few cases where self-training had both a lower mean squared error and a lower Spearman rank correlation (see 12-layer ESM-1 model with 85 million parameters trained on RL401_YEAST_Bolon2013-selection_coefficient).

3.3 Examining Spearman rank correlation progress over the course of self-training

We next sought to explore whether the predictive power of self-training is a monotonic function of the number of iterations, whether the models increase in predictive power over the course of self-training, and how changing the number of maximum self-training iterations affects performance. These questions can all be addressed by observing the training and validation losses over time. We present these results in Figs. 3.5, 3.6, and 3.7.

The Spearman correlation trajectories show that the improvement provided by self-training is not always a monotonic function of the number of self-training iterations. On the contrary, there are a number of experiments where the validation Spearman correlation is either flat or follows a cyclic pattern. The latter case illustrates the importance of using early stopping to determine the number of self-training iterations.

In some instances the model seemingly learns nothing at all, as the training and validation curves are completely flat. Experiments training the 6-layer ESM-2 model with 8 million parameters have the highest proportion of such runs (Fig. 3.5) and this proportion decreases as the number of parameters in the model increases. This suggests that the experiments where no learning occurred are due to lacking model capacity - the model was not capable of capturing the complexity of the data. This is further supported when examining runs involving the 12-layer ESM-1 model with 85 million parameters. For any experiment in which it does not seem to learn, the corresponding experiments with the lower capacity models either follow the same (lack of) a trajectory or they learn and ultimately achieve a validation Spearman correlation that is lower than the correlation that the 85 million parameter model starts (and stays) at. It is also possible that this effect is due to the particular training and validation sets corresponding to the random seed. Future experiments can control for this variable by averaging results over multiple random seeds.

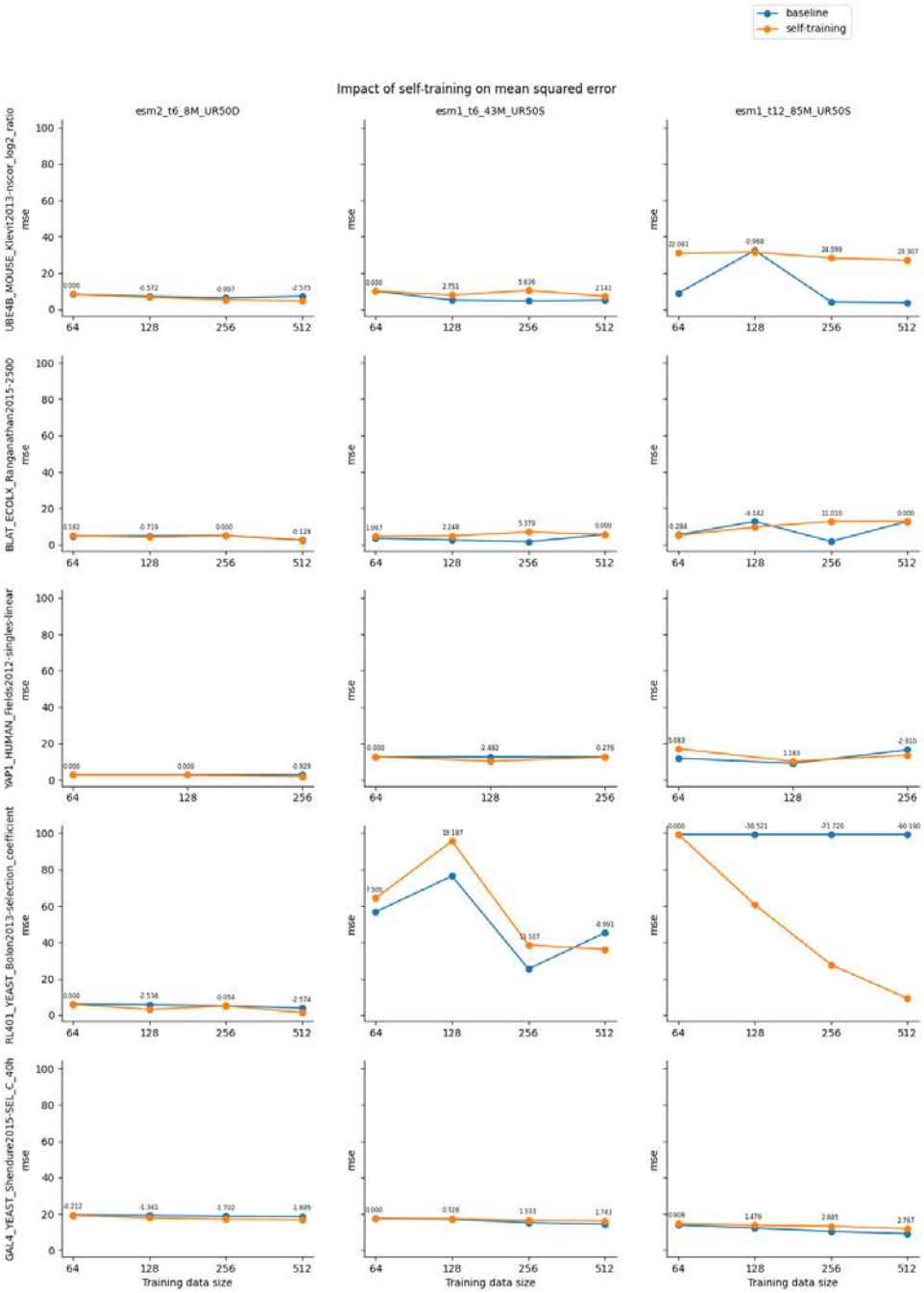

Figure 3.4: Comparison of mean squared error when applying self-training and standard fine-tuning. Model strength/complexity increases from left to right, and the number of unlabelled sequences increases from top to bottom. On every plot, each point is annotated with $MSE_{self-training} - MSE_{baseline}$.

Another interesting aspect revealed by these plots is the seemingly low correlation between validation Spearman correlation and test Spearman correlation. In order to perform this analysis, we consider only those experiments with validation Spearman correlations that have meaningful (i.e. not flat) trajectories. We expected that whenever the self-training validation Spearman correlation would be lower than the baseline's test Spearman correlation, the self-training test Spearman correlation would also be lower. This assumption ended up being correct for multiple experiments. However, there were a number of experiments where the self-training validation Spearman correlation was markedly higher than the baseline's test Spearman correlation and yet the self-training test Spearman correlation was at or below the baseline. This suggests that performing early stopping based on Spearman correlation may not be ideal. Further experiments could be performed in which the pseudo-log likelihood metric used in model training is also used for early stopping, in contrast to this work's use of the Spearman correlation for early stopping.

3.4 Examining the impact of unlabelled data magnitude

A major decision in this work's experimental design was using multiple sequence alignments as the source of unlabelled data. This decision was largely driven by the requirement that the number of unlabelled sequences should be significantly larger than the number of labelled sequences ($M << N$ [10]). This decision may have resulted in the use of sequences that were from a different distribution than the labelled data. Fig. 3.8 illustrates how the magnitude of unlabelled data correlated with the impact of self-training. While there is no correlation with the smallest ESM model, there is a slightly negative correlation with the 6-layer ESM-1 model with 43 million parameters and a positive correlation with the largest model. It should also be emphasized that despite the positive correlation in the final case, the regression line captures mostly negative improvement due to self-training. However, the positive correlation serves as evidence that with a higher ratio of unlabelled to labelled data or a more complex model (or both), self-training would lead to more impressive results.

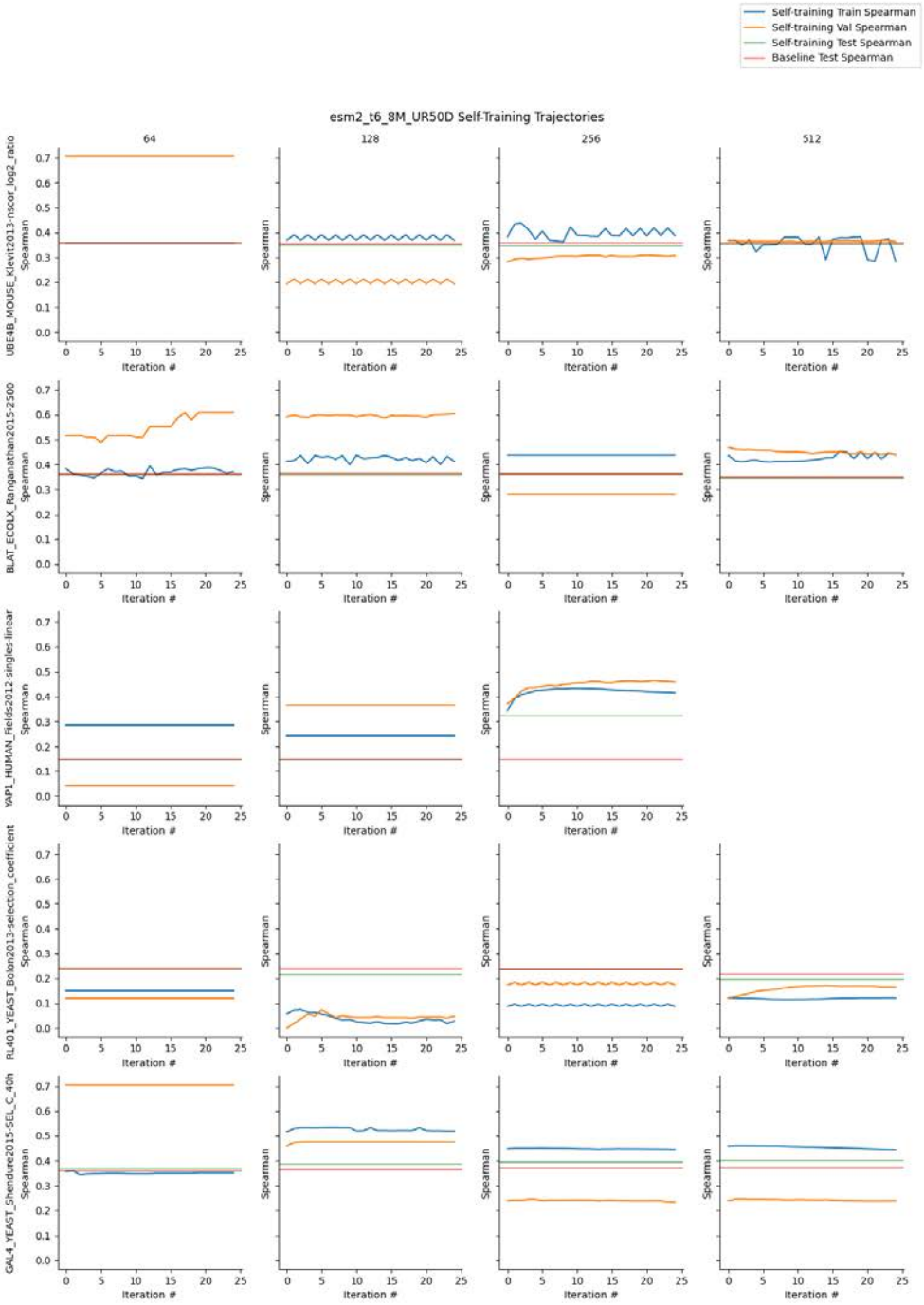

Figure 3.5: Change in Spearman rank correlation over the course of self-training for the 6-layer ESM-2 model with 8 million parameters. The number of labelled sequences used from training increases from left to right. The number of unlabelled sequences available in the dataset increases from top to bottom.

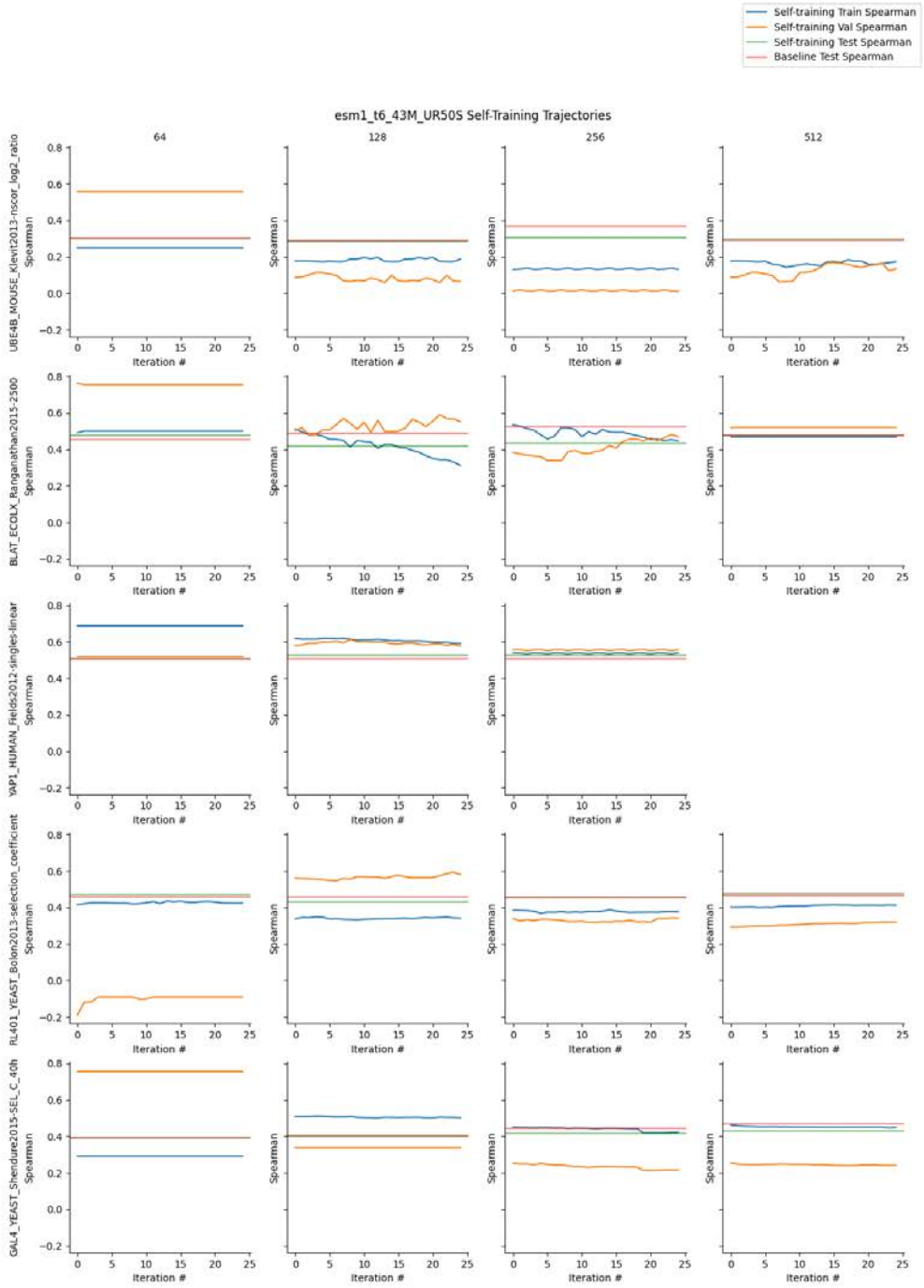

Figure 3.6: Change in Spearman rank correlation over the course of self-training for the 6-layer ESM-1 model with 43 million parameters. The number of labelled sequences used from training increases from left to right. The number of unlabelled sequences available in the dataset increases from top to bottom.

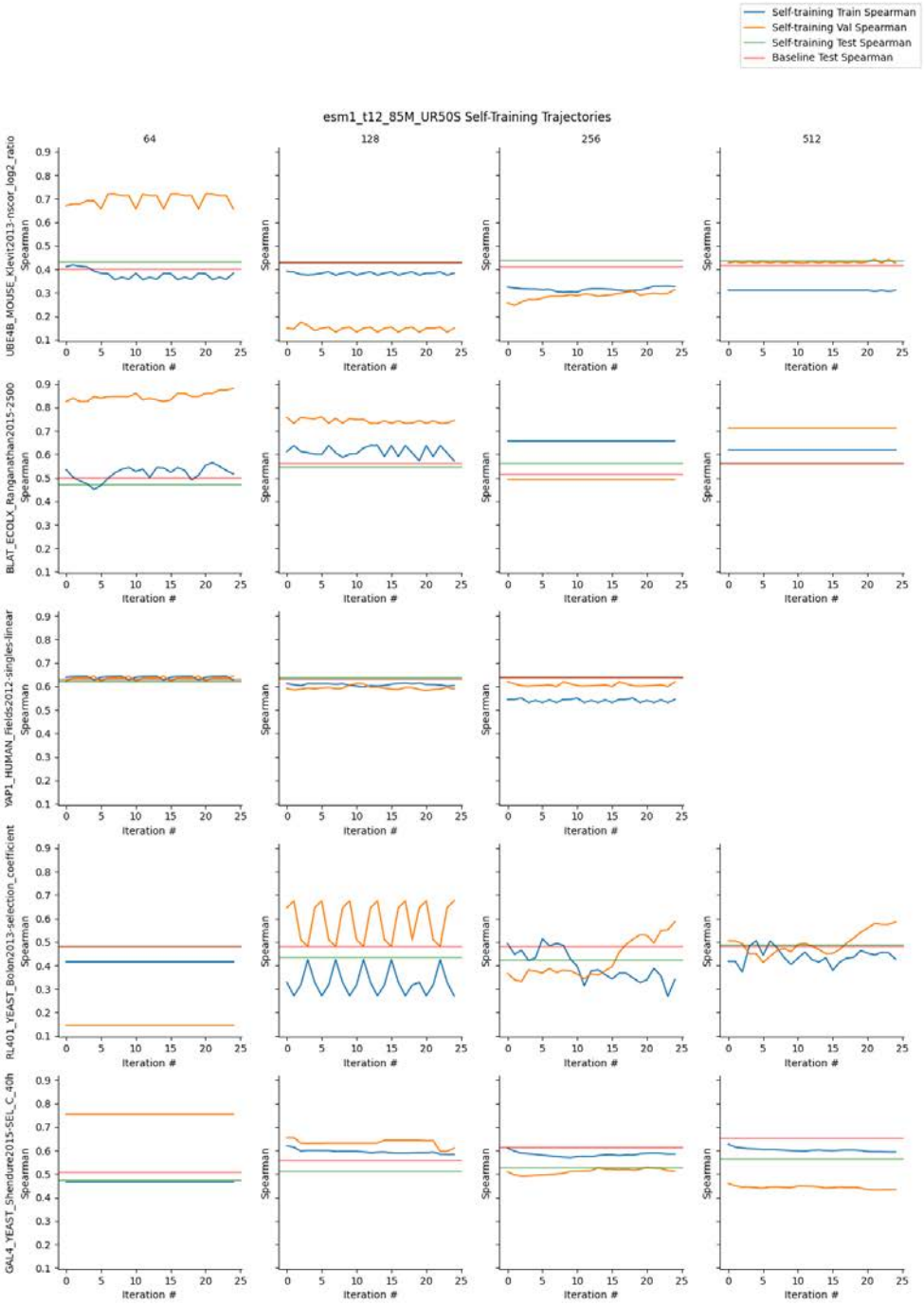

Figure 3.7: Change in Spearman rank correlation over the course of self-training for the 12-layer ESM-1 model with 85 million parameters. The number of labelled sequences used from training increases from left to right. The number of unlabelled sequences available in the dataset increases from top to bottom.

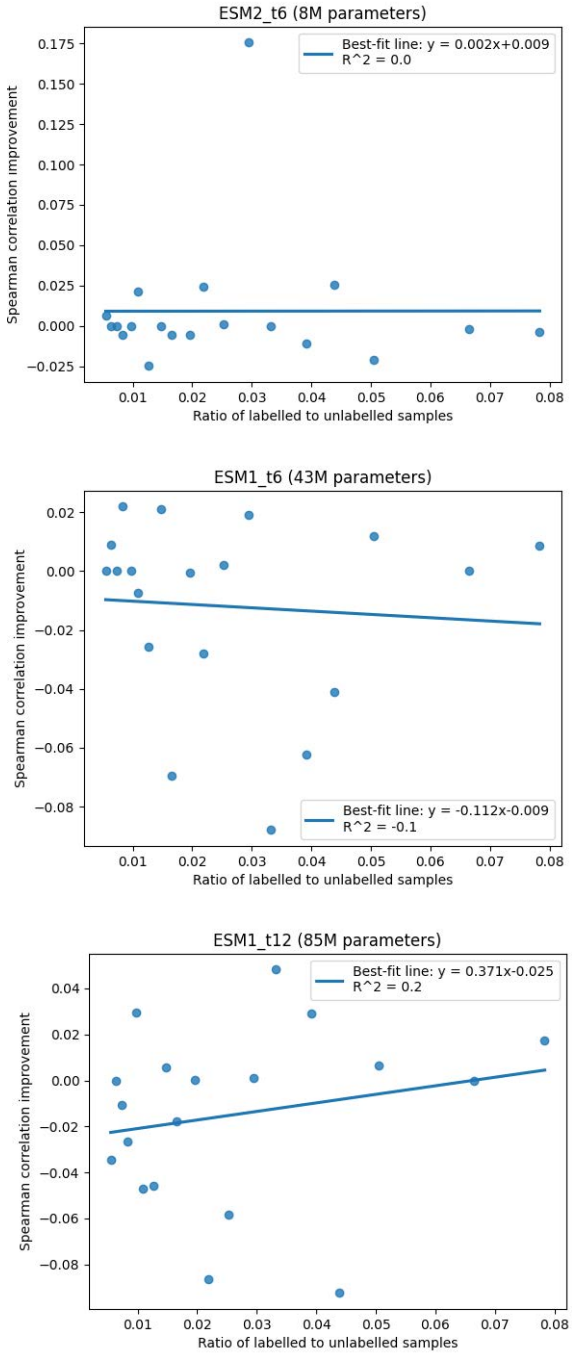

Figure 3.8: Linear regression to determine the correlation between magnitude of unlabelled data and difference in Spearman rank correlation produced by self-training. Model complexity increases from top to bottom.

Chapter 4

Conclusion

Overall, our initial tests of applying self-training to protein fitness prediction are underwhelming. Performance increases compared to standard fine-tuning are uncommon, and when they are present, they are modest. The computational cost of applying self-training is many times that of standard fine-tuning, as self-training is essentially the repeated applications of a modified form of fine-tuning. Increased computational costs come also from the fact that this modified form of fine-tuning takes in a much greater magnitude of data as input ($M << N$). Therefore, the marginal utility of these modest performance gains is extremely low. We note that these results are a reflection of the particular choices made in designing the experiments (e.g. number of self-training iterations, number of fine-tuning iterations, models used) for this project. Changes in experimental design choices, such as those outlined below, may improve downstream results.

4.1 Future Work

Increasing time and compute

With less limitations on time and computational power, experiments could be run with multiple random seeds, which would provide for replicability and further confidence in the results. In addition, this would also allow us to design experiments with parameters that are closer to the experiments in the STraTA paper[10]. Specifically, it would allow us to run self-training with more self-training iterations, more fine-tuning iterations, larger models (such as ESM-1b), and larger datasets whose sizes are closer to the ones used there.

Random restarts

The authors of the STraTA paper mention that they "perform 10 random restarts where there are less than 10K training examples and report the mean and standard deviation"[10]. They mention that this choice is made since "fine-tuning BERT can be unstable on small datasets"[10]. This method could potentially benefit every experiment included in this project, as the datasets trained upon were chosen to minimize the number of training samples as a result of compute limitations. However, since the models we chose were markedly smaller than BERT, it's also possible that the instability that applies to BERT would not apply to these experiments. This ambiguity caused by opposing factors makes the use of random restarts a promising direction for future research.

Intermediate fine-tuning

Previous work by He et al. also mentions how performing another step of fine-tuning on only the labelled dataset after fine-tuning on the combined true labels and pseudolabels improves performance [3, 10]. Due to compute limitations, we opted to forgo this step in this work's experiments. However, including it (either indiscriminately for all runs or selectively based on validation data) within the self-training algorithm could yield better results. Drawing an analogy from the mechanism coined by Biswas et. al., intermediate fine-tuning could potentially serve to guide the model's knowledge towards sequences that more closely mirror the wild-type sequence of the labelled dataset being fine-tuned upon [1].

Distributions of unlabelled sequences

Within this work, MSA sequences were used as unlabelled sequences. Other options included discarding labels for assay-labelled sequences to create a set of "unlabelled" sequences, artificially inserting single mutations into the wild-type sequences, or some combination of the above approaches. The MSA sequences were chosen for these experiments for two reasons - they were known to be naturally occurring and having evolutionary history with the wild-type sequence and the number of MSA sequences fulfilled the "$M << N$" relation between the labelled and unlabelled datasets.

The STraTA paper mentions the importance of using unlabelled data that is of the same distribution as the labelled data, particularly stating how "using OOD (out of domain/distribution) unlabeled examples typically leads to worse self-training results compared to using in-domain unlabeled examples except for [one case], and combining the two types of unlabeled examples does not bring further improvements..."

[10]. A further avenue of exploration of how to apply these findings to proteins has the potential to yield better self-training results but also a greater understanding of what it means for protein sequences to be "of the same distribution".

Biological task augmentation

A major direction for future research comes from the title of the STraTA paper itself - task augmentation in a biological context. In the words of the authors, "task augmentation exploits unlabeled texts from the domain of a given target task to simulate a large amount of in-domain training data for [an] auxiliary task... which is then used to train a given model before applying it to the target task" [10]. This direction for future work naturally follows from the previous suggestion, as both directions relate to the distributions of protein data. The results of previous work clearly indicate that using task augmentation and self-training in tandem result in greatly increased performance. However, applying this two-pronged approach to protein fitness prediction would require first determining a suitable auxiliary task.

STraTA's Low Resource Setting

One major way in which this work differs from the approach taken by STraTA is in the lack of a development set. For this project, we opted to split the labelled data into training and validation sets. This means that in all reported results, only a fraction of the reported n was used for training. This experimental design decision was made in an attempt to remain true to the spirit of few-shot learning. In contrast, the authors of the STraTA paper used all of their reported n for training, using a separate development set for early stopping and validation purposes. The authors acknowledge that using a development set is infeasible in real-world low-resource scenarios, and propose an alternate approach in which they fine-tune each model for a fixed number of 512 steps, checkpoint every 30 steps, and evaluate a single model obtained by averaging the last 5 model checkpoints. They use a fixed number of 30 self-training iterations, each following the previously described same fine-tuning procedure [10]. The context to which this modified self-training algorithm applies closely matches the real-world scenario regarding protein assay labels to which this work applies. Therefore, applying this modified self-training algorithm could lead to increased performance.

Filtering pseudolabels

The authors of STraTA attribute the use of a broad distribution of pseudolabels as a major factor in the impressive performance gains provided by self-training. While earlier self-training approaches only included pseudolabels that the model was sufficiently confident about [10], STraTA's approach differs in that the authors opt to perform no filtering on the pseudolabels, creating a broad distribution for the model to train on. This decision is rooted in a previous finding that "large language models like BERT are overconfident and poorly calibrated" [6, 10]. However, as neither of these works deal with protein sequences, it is possible that ESM models are not susceptible to the same issue. Therefore, filtering pseudolabels by model confidence may lead to improved performance in the case of self-training for protein fitness inference.

Using benchmark tasks from FLIP

Finally, it would be informative to apply self-training to the benchmark tasks described by Dallago et. al. in the FLIP paper [2]. This would allow for further evaluation of self-training on standardized benchmark tasks that utilize publicly available data.

Bibliography

[1] Surojit Biswas et al. "Low-N protein engineering with data-efficient deep learning". In: *Nature methods* 18.4 (2021), pp. 389–396.

[2] Christian Dallago et al. "FLIP: Benchmark tasks in fitness landscape inference for proteins". In: *bioRxiv* (2021), pp. 2021–11.

[3] Xuanli He et al. "Generate, annotate, and learn: Nlp with synthetic text". In: *arXiv preprint arXiv:2106.06168* (2021).

[4] Chloe Hsu et al. "Learning protein fitness models from evolutionary and assay-labeled data". In: *Nature biotechnology* 40.7 (2022), pp. 1114–1122.

[5] *Introduction to Information Retrieval - Evaluation*. `https://web.stanford.edu/class/cs276/handouts/EvaluationNew-handout-6-per.pdf`. Apr. 2013.

[6] Zhengbao Jiang et al. "How can we know what language models know?" In: *Transactions of the Association for Computational Linguistics* 8 (2020), pp. 423–438.

[7] Facebook AI Research. *Evolutionary Scale Modeling*. `https://github.com/facebookresearch/esm`.

[8] Adam J Riesselman, John B Ingraham, and Debora S Marks. "Deep generative models of genetic variation capture the effects of mutations". In: *Nature methods* 15.10 (2018), pp. 816–822.

[9] Alexander Rives et al. "Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences". In: *Proceedings of the National Academy of Sciences* 118.15 (2021), e2016239118.

[10] Tu Vu et al. "Strata: Self-training with task augmentation for better few-shot learning". In: *arXiv preprint arXiv:2109.06270* (2021).