Extrapolative Controlled Sequence Generation via Iterative Refinement

Vishakh Padmakumar [1] **Richard Yuanzhe Pang** [1] **He He** [1] **Ankur P. Parikh** [2]

Abstract

We study the problem of extrapolative controlled generation, i.e., generating sequences with attribute values beyond the range seen in training. This task is of significant importance in automated design, especially drug discovery, where the goal is to design novel proteins that are *better* (e.g., more stable) than existing sequences. Thus, by definition, the target sequences and their attribute values are out of the training distribution, posing challenges to existing methods that aim to directly generate the target sequence. Instead, in this work, we propose *Iterative Controlled Extrapolation* (*ICE*) which iteratively makes local edits to a sequence to enable extrapolation. Specifically, we train the model on synthetically generated sequence pairs that demonstrate small improvement in the attribute value. Results on one natural language task (sentiment analysis) and two protein engineering tasks (ACE2 stability and AAV fitness) show that *ICE* considerably outperforms state-of-the-art approaches despite its simplicity.[1]

1. Introduction

Controlled generation, i.e., generating sequences $\boldsymbol{x}$ with a desired attribute z, is a pervasive problem across multiple domains. In natural language processing (NLP), z could represent the sentiment or the style (e.g., formality) of a sentence. In computational biology, z could represent the stability, fluorescence, binding affinity, or other properties of a protein sequence.

Occasionally, abundant supervised data of the form $(\boldsymbol{x}, z)$ exist, such as Wikipedia domains or Gene Ontology categories (Keskar et al., 2019; Madani et al., 2020), enabling direct training of a conditional generation model $p(\boldsymbol{x}|z)$. In cases where the amount of supervised pairs available is small, it is typical to train a scorer $f(\boldsymbol{x})$ on this data, which maps from an input sequence to an output attribute value. One can then use $f(\boldsymbol{x})$ to annotate a large corpus for training (Gehman et al., 2020) or directly use $f(\boldsymbol{x})$ during inference to guide the generation process of an unconditional model $p(\boldsymbol{x})$ (Dathathri et al., 2020; Yang & Klein, 2021).

In this work, we focus on applications where it is necessary to generate sequences with attribute values that *extrapolate* beyond the training distribution. For example, in biological sequence design, the problem of generating *de novo* (novel) sequences that are *better* than existing natural sequences with respect to some attribute (e.g., binding affinity to a specific target) is of critical importance to drug discovery (Arnold, 1998; Romero & Arnold, 2009; Freschlin et al., 2022). In creative text generation, we want to generate text that accentuates a stylistic attribute (e.g., humor) beyond simply imitating existing literature (He et al., 2019; Lyu et al., 2021).

Existing controlled generation paradigms often extrapolate poorly when the range of attribute values z in the training data has limited coverage, as both $p(\boldsymbol{x}|z)$ and the attribute scorer $f(\boldsymbol{x})$ may not generalize outside of the training range of the attribute. For example, consider the ACE2 stability task (Chan et al., 2021c) shown in Figure 1, where the goal is to generate mutants of the ACE2 protein that have higher stability (lower *ddG* value). The training data contains sequences with *ddG* values varying between -4 and 10, but during inference, we want to generate more stable proteins than what we already have, e.g., extrapolate to *ddG* less than -5. Since this range of z is not supported on the training data, directly fitting $p(\boldsymbol{x}|z)$ to the training data will result in unpredictable performance for $z < -4$.

Our main assumption is that even though sequences with different target values, such as stable and unstable proteins, have distinct distributions, the process of transforming one sequence into a *slightly improved* version is applicable to different ranges of attribute values. For instance, in drug design, better proteins are often achieved by evolving from successive mutants, and in text generation, the sentiment can be strengthened by adding adverbs of degree. Therefore, we propose to the problem into a series of local improve-

[1]New York University [2]Google DeepMind. Correspondence to: Vishakh Padmakumar <vishakh@nyu.edu>.

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

[1]Our code and models are available at https://github.com/vishakhpk/iter-extrapolation.

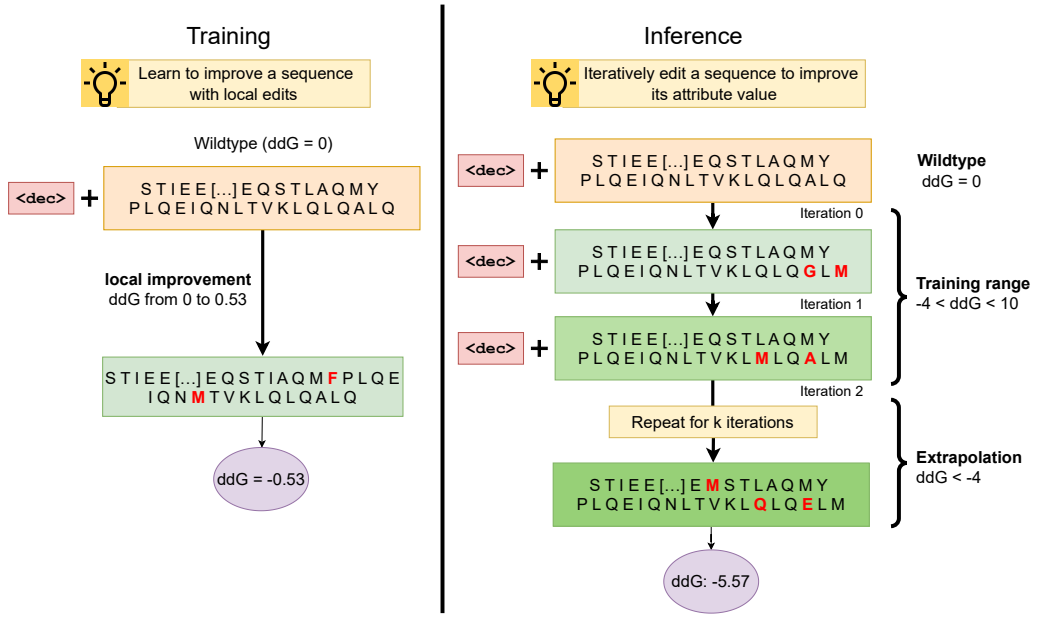


Figure 1: An overview of the approach, *Iterative Controlled Extrapolation* (*ICE*), on the ACE2 stability task. Our initial dataset only contains proteins with ddG values (lower means more stable) between -4 and 10. During training, we generate perturbations of protein sequences and learn a generator to make local edits of a base sequence to reduce its ddG value. At inference time, we iteratively apply the trained generator which achieves a ddG value of -5.57 after 10 iterations, more stable than the mutations seen during training.

ments made to a base sequence $\boldsymbol{x}_0$. Our intuition is that this local improvement is stable across attribute values. Thus we can learn these local edits (or mutants) on the training distribution and apply it in succession at inference time to extrapolate to new ranges of attribute values.[2]

As shown in Figure 1, to train the local editor, we synthetically generate close pairs of sequences using a masked language model (Devlin et al., 2019), such that they differ marginally in attribute values. During inference, our model uses two control tags, <inc> for increment and <dec> for decrement, to locally improve a sequence in the desired direction. Increasing the number of edits on the sequence enables extrapolation. We call our approach *Iterative Controlled Extrapolation* (*ICE*).

We evaluate our approach in both the natural language and protein domains. For text generation, we generate reviews with a sentiment either more positive or negative than seen in the training data. For protein engineering, we present results on two tasks—generating mutations of the ACE2 protein that have higher stability measured by FoldX (Schymkowitz et al., 2005) and generating mutations of an adeno-associated virus (AAV) capsid protein (Bryant et al., 2021) with a higher fitness value. *ICE* achieves consistent extrapolation on these three tasks, outperforming both standard methods for controlled generation such as PPLM (Dathathri et al., 2020) and a state-of-the-art extrapolative controlled generation method, Genhance (Chan et al., 2021a). In particular, in the AAV task, despite seeing zero sequences that are better than the wildtype AAV sequence during training, our model is able to generate a diverse range of better candidates as judged by an oracle model.

[2]These iterative improvements are internal to our model and thus not analogous to rounds in directed evolution (Arnold, 1998), which typically require access to a wet lab experiment (or oracle) after each round.

2. Related Work

2.1. Controlled Generation

While controlled generation has been studied extensively in the literature, most of these methods do not focus on the extrapolation setting. We present an overview here situating our method and setup amongst prior work.

Methods using control codes Keskar et al. (2019) and Madani et al. (2020; 2023) learn a conditional sequence model $p(\boldsymbol{x}|c)$ where c is the control code, encoding either a discrete or scalar value specifying the target attribute. However, these models may struggle when conditioning on unseen attribute values outside the training data range. Instead of conditioning on absolute target values, Lu et al. (2022) attempt to overcome this limitation by sampling generations from a model, iteratively quantizing these into

more fine-grained control codes and then using the highest bucket for controlled generation.

Iterative editing methods Our approach is also related to edit-based approaches (Guu et al., 2018; Mallinson et al., 2022; Novak et al., 2016), and closely connected to concurrent work, Welleck et al. (2023), that samples and scores generations from a model in order to learn edits in various NLP tasks. The key distinction to our work is that we focus on extrapolation. In the setup of Welleck et al. (2023), the model learns by seeking feedback on all generated pairs. However, we are explicitly interested in the case where the model is required to generate sequences outside the range where it is able to obtain feedback.

Latent variable models Another approach to achieve control is to model the attribute as a latent variable (Mueller et al., 2017; Gligorijević et al., 2021; Chan et al., 2021a;b). For example, Genhance (Chan et al., 2021a) proposes to represent the latent vector as a sum of attribute-relevant and attribute-irrelevant components. They then perturb the former to achieve extrapolation with applications to both NLP and biology. However, latent variable models on discrete sequence data are known to suffer from stability issues. In contrast, our approach makes edits in the text space, bypassing the problem of mapping from continuous latent spaces to discrete sequences.

Attribute control via a scorer model Another line of work (Dathathri et al., 2020; Yang & Klein, 2021; Li et al., 2022) adds attribute information via a scorer model $p(z|\boldsymbol{x})$ to guide an unconditional language model $p(\boldsymbol{x})$ at inference time. Because this approach heavily relies on the scorer model which is a trained classifier, it is not often conducive to extrapolation beyond the training data distribution, as we will show in our experiments. Alternatively, one could use the classifier as a reward model for reinforcement learning (Gong et al., 2019; Angermueller et al., 2020b) which suffers from similar shortcomings as the generator can exploit and amplify imperfections in the reward (Amodei et al., 2016; Ibarz et al., 2018; Pang et al., 2022).

2.2. Biological Sequence Design

The problem of generating *de novo* sequences that improve upon natural sequences is of massive value to drug discovery, healthcare, and agriculture, as signified by the 2018 Nobel Prize in Chemistry on directed evolution (Arnold, 1998). As a result, there has been a growing interest in using machine learning for this problem (Yang et al., 2019; Angermueller et al., 2020a; Freschlin et al., 2022; Ren et al., 2022). Brookes et al. (2019) tackle extrapolation via a series of importance sampling distributions, in contrast to our controlled generation approach.

The iterative nature of *ICE* is internal to our modeling approach and thus not analogous to rounds in directed evolution which typically require access to an oracle (or wet lab experiment) after each round. Rather, at each round of directed evolution, *ICE* could potentially be (iteratively) run and its final output interpreted as the proposed candidates for validation.

Generating and experimentally validating novel sequences from large pretrained protein language models is also an exciting but nascent area. These approaches (Madani et al., 2021; Verkuil et al., 2022) typically generate sequences by conditioning on broad categories or backbone structures, rather than optimizing towards a specific target attribute (e.g., stability or fluorescence) as we seek to do.

3. Our Approach

Problem setup We denote an input sequence with ℓ tokens as $\boldsymbol{x} = (x_1, ..., x_\ell)$ and an attribute value as $z \in \mathbb{R}$. Here $\boldsymbol{x}$ can represent a protein sequence of ℓ amino acids, where z represents its stability, or a textual restaurant review of ℓ tokens, where z corresponds to the associated sentiment score. During training, we are typically given a large unsupervised corpus $\mathcal{D}_{\text{unsup}} = \{\boldsymbol{x}^{(m)}\}_{m=1}^{M_{\text{unsup}}}$ of size M_{unsup} and a much smaller supervised corpus of sequences paired with attribute values, $\mathcal{D}_{\text{sup-train}} = \{(\boldsymbol{x}^{(m)}, z^{(m)})\}_{m=1}^{M_{\text{sup-train}}}$ of size $M_{\text{sup-train}}$. Let α_- and α_+ denote the lower and upper bound of z in $\mathcal{D}_{\text{sup-train}}$ respectively, i.e., $z \in [\alpha_-, \alpha_+]$ for all z in the training examples. We refer to this region as the *training region* of scores.

Our goal is to generate sequences that have an attribute value greater than (or less than) a target attribute value z^*. In particular, we aim to extrapolate beyond the training region, i.e., $z^* < \alpha_-$ or $z^* > \alpha_+$ depending on the application. We refer to these regions as the *extrapolation region* of scores.

Further, we assume that we have access to a scorer f_s that is trained on $\mathcal{D}_{\text{sup-train}}$ to predict the attribute value of each sequence, i.e., $\hat{z} = f_s(\boldsymbol{x})$. While f_s may achieve high performance on the training region of z, it is not trained on data from the extrapolation region and hence it can perform poorly when scoring examples in this range. Thus f_s should *not* be regarded as an oracle.

3.1. Overview

The core component of *ICE* is a local editor that modifies a short span within a sequence to improve its attribute value. Specifically, it takes in an input sequence $\boldsymbol{x}$ and a control token c that specifies whether to increase ($c =$ <inc>) or decrease ($c =$ <dec>) the attribute value, and outputs an improved sequence $\tilde{\boldsymbol{x}}$. We model the local editor $p_\theta(\tilde{\boldsymbol{x}} \mid \boldsymbol{x}, c)$ using a Transformer encoder-decoder model (Vaswani

et al., 2017). We train the editor by synthesizing pairs of sequences with a small difference in attribute value using masked language modeling (Section 3.2).

At inference time, starting with an initial sequence $\boldsymbol{x}_0$, we edit it iteratively until some stopping criteria is reached. Specifically, in iteration k, we edit the current sequence $\boldsymbol{x}_k$ to produce $\boldsymbol{x}_{k+1}$ by:

$$\boldsymbol{x}_{k+1} \sim p_\theta(\cdot \mid \boldsymbol{x}_k, c) \tag{1}$$

Each iteration is expected to move the attribute value of $\boldsymbol{x}_k$ toward z^*. We explore different ways of selecting the best candidate at each step of the inference as well as the stopping criteria of the inference process in Section 3.3.

3.2. Learning Local Edits from Perturbations

To train the local editor, we perturb examples from $\mathcal{D}_{\text{sup-train}}$ to generate training pairs with a small improvement toward the target value.

Specifically, given a sequence from the training region, $\mathcal{D}_{\text{sup-train}}$, we mask random tokens in it,[3] and use a masked language model to infill these to produce its perturbation (Figure 1). The masked language model is trained on the unsupervised data $\mathcal{D}_{\text{unsup}}$ such that the infill produces a valid sequence. To ensure that we make only small improvements, we predict the attribute value of each sequence using the scorer f_s, and retain only those pairs where the absolute difference in the attribute value is below a threshold δ.

Each pair of the original sequence and its perturbation gives us two examples for the editor: generating the perturbed sequence from the original sequence, and vice versa. Recall that the editor also takes in a control token that specifies whether the edit should increase or decrease the attribute value. For each input-output pair, we set the control code to be `<inc>` if the attribute value of the input sequence is less than that of the output sequence measured by the scorer f_s, and `<dec>` otherwise.

Given tuples of the input sequence, the output sequence, and the control code, we then train the editor p_θ on this dataset.

3.3. Inference

At inference time, we run the editor iteratively as described in Eq. (1).

Decoding method During each iteration, we experiment with two different ways in which to select the best candidate out of a set of generated sequences:

- **Scorer-free generation**: At each iteration of Equation (1), we perform generation using beam search relying on the *ICE* model likelihood to control the generation process.
- **Scorer-guided generation**: At each iteration, we generate a set of sequences via top-k sampling, score these with f_s and select the sequence assigned the highest (or lowest) score depending on the desired target value. While f_s is reliable in the training region, it is unclear if the guidance provided is beneficial to the *ICE* model as it generates sequences having attribute value in the extrapolation region.

Stopping criteria The objective of the task is to edit the input sequence to have an attribute value greater than (or less than) the target value z^*. However, reliably identifying when the inference process has reached z^* is difficult as it lies in the extrapolation region. In this work, we run inference for a constant number of iterations. We include additional discussion on the stopping condition in Appendix C.4.

4. Experimental Setup

We evaluate our approach on one NLP task and two protein design tasks—sentiment controlled generation (Section 5), the ACE2 stability task (Section 6), and the AAV fitness task (Section 7).

4.1. Evaluation

We are interested in measuring the ability of a model to successfully edit a sequence to have an attribute value greater than (or lesser than) a target value z^*. In our experiments, we report the success rate or the fraction of sequences that the model is able to edit to meet this criterion as determined by an oracle model. The oracle varies based on the task and is detailed in each of the experiment sections (Section 5, Section 6, Section 7).

4.2. Baselines

We benchmark the performance of our method against the following baselines. (a) **Sampling**: A simple baseline is to directly edit sequences using a masked language model. Mirroring the synthetic data creation process from Section 3.2, we mask and infill a random span within the initial sequence to change its attribute value. (b) **Iterative Sampling**: To ablate the contribution of the editor model in *ICE*, we replace it with a mask-and-infill editor using a masked language model; the rest of the iterative algorithm is the same as *ICE* with the *Scorer-Guided* inference method. (c) **Genhance**: We compare to *Genhance* (Chan et al., 2021a), an extrapolative baseline which performs controlled generation by making perturbations in a latent space learned to encode the attribute value. Increasing the size of these perturbations during inference enables extrapolation.

[3]The specific masking strategy varies depending on the task and is specified in each of the experiment sections (Section 5, Section 6, Section 7).

For the NLP task, we compare to two additional baselines. (d) **PPLM** (Dathathri et al., 2020) is a controlled generation method that guides the generation of an autoregressive language model at inference time using a scorer, $p(z|\boldsymbol{x})$. We use f_s as the scorer to guide the generation. We include the baseline to evaluate if the guidance from the scorer trained on the training region allows for extrapolation. (e) **Score-Conditioned Generator**: We also compare to a score-conditioned model, which generates the output sequence given the input and the target attribute value.[4] To train the score-conditioned model, we use the same synthetic data (Section 3.2) but replace the control code with the attribute value of the output sequence measured by f_s appended as a string token. At inference time, we append the desired target score and evaluate if the model generalizes to the unseen score values.[5]

5. Sentiment Control

In this task, the objective is to control the sentiment associated with a short paragraph of text (2–3 sentences). We use the Yelp dataset for this task (Zhang et al., 2015), which consists of 650K training examples and 50K test examples, evenly divided into sentiment scores from 1 to 5. We define the *training region* as the range of sentiment scores from 2 to 4 and the *extrapolation region* as the range of scores from 1 to 2 and 4 to 5. For this task, we are interested in measuring the ability of the model to extrapolate in both directions, i.e., increase and decrease the associated sentiment of an example. To measure this, we report the success rate of editing the sentiment beyond the following *target values*—1.5 and 2.5 in the negative direction and 3.5 and 4.5 in the positive direction. 1.5 and 4.5 belong to the *extrapolation region*.

5.1. Implementation Details

Training the scorer We fine-tune a RoBERTa-Large model (Liu et al., 2019) on the examples from the Yelp dataset in the *training region* to serve as the scorer, f_s. The scorer is a regression model that takes in the input text and predicts its sentiment score, a real number between 2 and 4. Appendix B describes further training details of the scorer.

Training the editor To create the synthetic data through perturbation, we mask tokens using the strategy described in Lewis et al. (2020) and infill these with a pre-trained BART-Large model.[6] We filter the pairs created by setting the hyperparameter $\delta = 0.4$ (Section 3.2). We fine-tune the T5-Base model (Raffel et al., 2022) on the synthetic training data to obtain the local editor. Appendix B describes further training details.

Inference We run inference using both methods described in Section 3.3. For scorer-free inference, we use beam search with a beam size of 5. When performing scorer-guided inference, at each iteration, we generate 5 sequences using top-k sampling with $k = 5$ and a temperature of 0.7; we then select the best one using f_s. We run 10 steps of iterative editing for both methods.

Evaluation We report results on a random subset of 1831 examples from the test set of the Yelp dataset against all 4 aforementioned targets.[7] To evaluate whether the attribute value of the final generated sequence extrapolates beyond the *training region*, we estimate the ground-truth sentiment scores via an oracle—a RoBERTa-Large model that is fine-tuned on the entire Yelp dataset, i.e., both the *training* and *extrapolation regions*.

Baselines For sentiment control, we compare our method to *Sampling*, *Iterative Sampling*, *Genhance*, *PPLM*, and the *Score-Conditioned Generator*. We use T5-Base to train the *Score-Conditioned Generator* to match the *ICE* editor. The architecture of the *Genhance* model is also based on T5-Base, making it comparable in size to *ICE* editor. At inference time, for each test example, we sample 50 sequences from *Genhance* and use f_s to select the best one to match the total number of sequences generated by *ICE* in all iterations. For *Iterative Sampling*, we generate 5 sequences per iteration for 10 iterations and use f_s to select the best one at each iteration, the same as *ICE*.

5.2. Results

ICE* outperforms the baselines in the *extrapolation region From Table 1, we see that the *ICE* model (when guided by the scorer) strongly outperforms the baseline methods in the *extrapolation region*. Even without the scorer, the *ICE* model achieves performance on par with the strongest baseline, *Genhance*. Table 7 in Appendix C.2 shows an example of increasing the sentiment associated with a sentence over multiple iterations.

[4] This baseline is similar to the methods described in Jain & Berg-Kirkpatrick (2021) and Chen et al. (2021).

[5] The score-conditioned baseline is trained on minimal edits and at test-time, we assess its ability to generalize to larger edits, which poses a challenge. Altering the training data to incorporate larger edits could improve the performance of this baseline however in our problem setting, we do not have pairs of sequences for the examples in $\mathcal{D}_{\text{sup-train}}$.

[6] The masking strategy involves sampling a location of the start of the span from a Bernoulli distribution ($p = 0.8$) and then selecting the number of tokens to mask by sampling from a truncated Poisson distribution ($\lambda = 6$). The maximum span size is set to 12. We report more variants of the masking strategy in Table 6 in Appendix C.2.

[7] We ensure that these examples are selected such that the sentiment value of the input text is within the *training region*.

Methods	Targets in Training Region			Targets in Extrapolation Region		
	3.5	2.5	Average	4.5	1.5	Average
Sampling	0.362	0.259	0.310	0.061	0.050	0.056
Iterative Sampling	0.668	0.657	0.663	0.320	0.328	0.324
Genhance	**0.982**	0.833	0.908	0.482	0.291	0.387
Score-Conditioned Model	0.780	0.766	0.773	0.212	0.217	0.215
PPLM	0.534	0.516	0.522	0.081	0.065	0.077
ICE Scorer-Free	0.976	**0.918**	**0.947**	0.446	0.305	0.376
ICE w/ Scorer	0.943	0.900	0.921	**0.638**	**0.582**	**0.610**

Table 1: Results on the sentiment control task. We report the success rate measured as the fraction of examples that have a sentiment value greater than (or less than) a target score as determined by the oracle. Bold values indicate the highest rates of extrapolation. *Iterative Sampling*, *Genhance*, and *PPLM* use the scorer for inference. *ICE* achieves the highest success rate in the extrapolation region compared to the baselines.

Scorer guidance is beneficial We observe that the scorer helps both the *Iterative Sampling* baseline and *ICE* in sentiment control. *Iterative Sampling* benefits from the scorer with extrapolation performance increasing to 32.4% from the 5.6% observed in *Sampling*. The *ICE* success rate when guided by the scorer goes up from 37.6% to 61.0%. We do observe that *PPLM* extrapolates poorly despite using the scorer f_s. This highlights that f_s could be more useful for guiding inference when used to rank generated sequences, as in *ICE* and *Iterative Sampling*, as opposed to the conditional probabilities from f_s being directly used to guide the generation, as in *PPLM*.

What does *ICE* do in each iteration? To analyze how the sentiment score of the text is changed over iterations, we plot the difference between the sentiment score of the output at each iteration and that of the initial sequence. We randomly sample 100 examples from the test set, and use *ICE* to increase their sentiment scores. We collect the output of *ICE* at every iteration using the scorer-free inference. We then plot the histogram of the increase in sentiment score (with respect to the initial score) for iterations 1, 4, 7, and 10 in Figure 2. As the iteration count increases, we observe that the increase in sentiment scores also becomes larger (i.e., the mode of the distribution is moving right), although the editing is not always successful (the scores of a small number of outputs decrease from the initial score and fall in the negative buckets). Overall, this shows that *ICE* is able to increase the sentiment score on average via iterative editing.

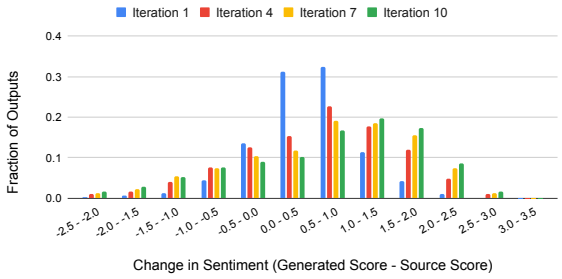


Figure 2: We plot the histogram of the increase in sentiment scores with respect to the initial score at every iteration of *ICE* on 100 examples. As the iteration count increases, we observe that the mode of the distribution moves towards the positive side, suggesting that more examples are edited to be increasingly positive, resulting in extrapolation eventually.

6. Protein Design on the ACE2 dataset

Developing ways that generate more stable proteins could benefit drug discovery, as these proteins could potentially allow easier storage and have more reliable clinical effects compared to the existing proteins (Wang, 1999; Shire et al., 2004; Bloom et al., 2006; Deller et al., 2016; Webber et al., 2016). The objective of this task is to generate mutants of the human angiotensin-converting enzyme 2 (ACE2) wild-type sequence[8] that have higher stability. The stability value of the mutants is measured using the change in free energy from the wild-type, or *ddG*, via FoldX (Schymkowitz et al., 2005).[9] The wild-type itself has a *ddG* value of zero and more negative values represent more stable mutants. This synthetic task was created in Chan et al. (2021a) and we replicate their setup. The proteins are represented by a sequence of 83 amino acids out of a vocabulary of 20 different amino acids. In order to enforce that the mutations do not diverge too widely from the wild-type, a constant span of 8 amino acids (NTNITEEN) is kept fixed in all mutations. We view the *training region* to be the range of *ddG* values from -4 to $+10$. The *extrapolation region* refers to *ddG* values below -4. For this task, we aim to generate mutants having more negative *ddG* values. We measure this by reporting the success rate of generating mutations having *ddG* below target values, z^*, in the *training region*, -1 and -2.5, and

[8] https://www.uniprot.org/uniprotkb/Q9BYF1/entry
[9] https://foldxsuite.crg.eu/

the *extrapolation region*, -5, -6, and -7.

6.1. Implementation Details

Training the scorer To train f_s we fine-tune ProtBert (Elnaggar et al., 2021) on the examples with *ddG* values in the *training region* from the dataset in Chan et al. (2021a).

Training the editor We create pairs of sequences using the mask-and-infill approach from Section 3.2 using a pre-trained Prot-T5-XL model (Elnaggar et al., 2021). We sample token masks from a Bernoulli distribution with $(p = 0.8)$. To filter small perturbations, we set δ to 1.5. We then fine-tune Prot-T5-XL on this data to serve as the *ICE* editor.

Inference At inference time, we start from the wild-type and generate mutations with and without the scorer, f_s (Section 3.3). When using the scorer, we sample 5 sequences at each step, select the best one using f_s, and repeat the process for 10 iterations. For scorer-free inference, we generate sequences with beam size of 5 for 10 iterations.[10]

Evaluation In the ACE2 task, we are interested in generating mutants that have a lower *ddG* value. So we generate $10,000$ mutants of the wild-type from each model and report the success rate of generating mutants that have a *ddG* value lower than each of the task targets using FoldX as the oracle. We match the FoldX evaluation parameters from Chan et al. (2021a) to evaluate the mutations. We also report the average score of the Top-100 and Top-1000 mutants as determined by the oracle to evaluate the quality of the top candidates in the library of 10,000 produced by each model.

Baselines We compare our approach against *Sampling*, *Iterative Sampling*, and *Genhance*.[11] For *Genhance*, we report results from the model released by Chan et al. (2021a) on $10,000$ mutants generated with and without the scorer. This model is based on Prot-T5-XL as well making it directly comparable to the *ICE* model. For *Iterative Sampling*, we generate 5 sequences per iteration for 10 iterations.

6.2. Results

***ICE* outperforms baselines on extrapolation** Table 2 shows that *ICE* consistently outperforms *Genhance*, *Sampling*, and *Iterative Sampling* on all extrapolation targets. In addition, from Table 3, we see that *ICE* achieves a lower average *ddG* on the Top-100 and Top-1000 sequences. Interestingly, while *Iterative Sampling* achieves higher extrapolation rates than *Genhance* (Table 2), *Genhance* achieves a better average score on the Top-1000 and Top-100 subsets (Table 3) indicating that *Genhance* produces a smaller number of slightly more stable mutants (though still outperformed by *ICE*).

The scorer is valuable for all models in ACE2 In this task, we begin the generation from the wild-type (*ddG* score of zero) and the scorer, f_s, reliably guides the generation process until the score of -5. As a result, we see that all the methods strongly benefit from using the scorer (Table 2). In Figure 3, we plot the histogram of scores of the generated mutations from *ICE* and the reported baselines. From Figure 3a, we see that the peaks of the distribution of scores for all models move in the negative direction to be centered closer to -5 as compared to Figure 3b highlighting the value of the scorer. We do however note that our approach is able to achieve some extrapolation even in the scorer-free regime, far outperforming *Sampling* and achieving extrapolation at a higher rate than *Genhance*.

7. Protein Design on the AAV dataset

The AAV dataset (Bryant et al., 2021) aims to study the fitness landscape of an adeno-associated virus (AAV) capsid protein that is a key component of gene therapy (Russell et al., 2017). Our goal is to obtain mutants of the AAV-2 wild type sequence[12] that have a higher fitness value. We use the splits proposed by the FLIP benchmark (Dallago et al., 2021) for our experiments. Each mutant is a sequence of length varying from 734 to 750. Mutations are made on the wild-type sequence between indices 561 and 588. We use the provided *low-vs-high* split of the dataset to demarcate the *training region* and *extrapolation region*. The *training region* corresponds to fitness values below zero and the *extrapolation region* corresponds to positive fitness values. At inference time, the generation process begins at the wild-type, with a fitness score of zero, and the model is expected to generate mutants that have a positive fitness score. We evaluate performance against target values, z^* in the *training region*, -1, and in the *extrapolation region*, 0, 1, and 2.

7.1. Implementation Details

Training the scorer The scorer, f_s, is a CNN model trained on the examples in the *training region*. The architecture and hyperparameters for the CNN were chosen based on the FLIP benchmark.[13] The scorer accepts a string corresponding to the proteins and outputs a floating-point

[10] We present further analysis on the variation in performance based on the hyperparameters of generation in Appendix C.3.

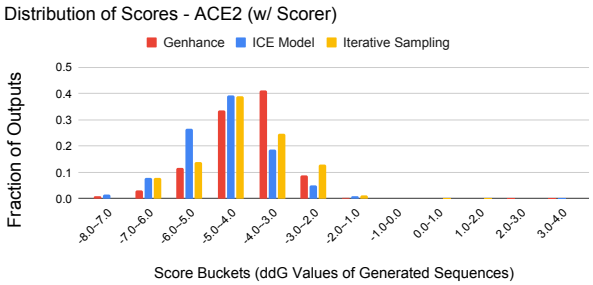
[11] The ACE2 task requires generating mutants of a specific wild-type. Pretrained autoregressive language models in the protein domain cannot generate mutants directly, only continuing sequences. As a result, sequence-to-sequence models are more appropriate for this task. Hence, *PPLM*, which relies on an autoregressive model, is not included as a baseline. Also, we do not include the *Score-Conditioned Generator* baseline as the vocabulary of Prot-T5-XL tokenizer solely consists of amino acids, thus it cannot accept the output score as a token along with the input.

[12] https://www.uniprot.org/uniprotkb/P03135/entry

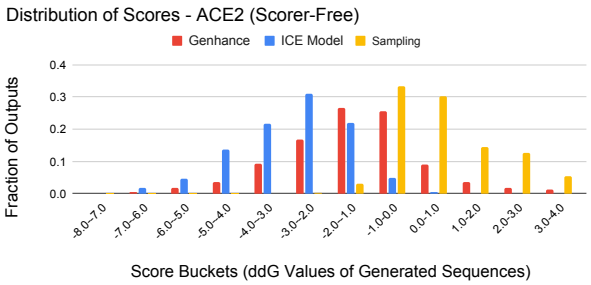
[13] On the low-vs-high split, the train correlation of the scorer is 0.82 and the test correlation is 0.34. This matches the best test correlation on this split obtained as part of the benchmark.

Methods	Targets in Training Region		Targets in Extrapolation Region		
	-1	-2.5	-5	-6	-7
Sampling	0.033	0.007	0.000	0.000	0.000
Iterative Sampling	0.998	0.954	0.220	0.079	0.001
Genhance Scorer-Free	0.570	0.219	0.021	0.005	0.001
Genhance w/ Scorer	**0.999**	**0.978**	0.159	0.040	0.009
ICE Scorer-Free	0.945	0.598	0.062	0.017	0.002
ICE w/ Scorer	0.998	0.974	**0.361**	**0.098**	**0.019**

Table 2: Results on the ACE2 task. The objective is to generate mutants of the wild-type that have higher stability i.e. lower *ddG* value. Each table cell represents the success rate of generating mutations lower than the corresponding target. Bold values indicate the highest rates of extrapolation. *ICE* achieves a higher rate of extrapolation than the reported baselines.



(a)



(b)

Figure 3: Histograms of *ddG* scores (lower is better) of the final mutations generated by *ICE* and the baselines on the ACE2 task. *ICE* generates higher quality mutations than the baselines both with (Figure 3a) and without the scorer (Figure 3b) guiding the inference. Further, the scorer significantly improves performance for all methods.

Library Size	Iterative Sampling	Genhance	*ICE*
All 10k	-4.326	-4.086	**-4.660**
Top 1k	-5.866	-6.030	**-6.575**
Top 100	-6.413	-7.354	**-7.938**

Table 3: Average *ddG* values (lower is better) of mutations generated from *Iterative Sampling*, *Genhance*, and *ICE* (each with the scorer). We report the average score of all 10000 mutations, the top 1000, and the top 100 as determined by the oracle. Bold values are the lowest average *ddG* value. *ICE* generates the most stable mutations.

fitness value.

Training the editor We create pairs to train the *ICE* model by following the same strategy as in ACE2. We use the Prot-T5-XL (Elnaggar et al., 2021) model to infill masks in the mutable region and score pairs with the scorer, f_s, to create the editor training data.[14] We then fine-tune Prot-T5-XL on this dataset. Since the length of the mutants is greater than the sequence length limit of Prot-T5-XL, we truncate them from the start to the last 512 tokens, which always contain the entire mutable region of the protein.

Inference We start from the wild-type and run inference on the *ICE* model as per Section 3.3. When using the scorer, we sample 5 generations, score them with f_s, select the best one, and repeat for 10 iterations. For the scorer-free setup, we generate with a beam size of 5 for 10 iterations.

Evaluation We generate $10,000$ mutants with each method and report the success rate of generating mutations that are above the target scores, z^*. In lieu of a wet-lab experiment, we obtain fitness scores for each generated sequence via an oracle model, which is a CNN trained on the *sampled* (i.i.d.) split of the AAV dataset.[15] This was chosen as the examples from the *sampled* split span fitness values across both the *training region* and *extrapolation region*.

Baselines We compare our approach to the *Sampling* and *Iterative Sampling* baselines.[16]

[14] We again set the hyperparameter δ to 1.5.

[15] We select the CNN architecture as it has the highest spearman correlation with the gold fitness values on the benchmark (Dallago et al., 2021). The model obtains a train spearman correlation of 0.93 and a test correlation of 0.92 on this split.

[16] As mentioned earlier, the *PPLM* and *Score-Conditioned Generator* baselines are not well suited for the protein tasks.

Methods	Targets in Training Region	Targets in Extrapolation Region		
	-1	0	1	2
Sampling	0.058	0.018	0.011	0.000
Iterative Sampling w/ Scorer	**0.524**	0.064	0.017	0.000
ICE Scorer-Free	0.481	0.188	0.033	0.001
ICE w/ Scorer	0.521	**0.223**	**0.036**	**0.002**

Table 4: Results on the AAV task. The objective is to generate mutations of the source protein that have a higher fitness value. We report the success rate of generating mutations with fitness values higher than the corresponding targets. Bold values indicate the highest extrapolation rates. *ICE* achieves a higher rate of extrapolation than the baselines.

Library Size	Sampling	Iterative Sampling	*ICE* Scorer-Free	*ICE* w/ Scorer
All 10k	-3.450	-1.390	-1.150	**-1.040**
Top 1k	-0.567	-0.584	0.403	**0.918**
Top 100	1.605	1.550	1.452	**1.750**

Table 5: Average fitness values (higher is better) of mutations generated from *Sampling*, *Iterative Sampling*, and *ICE*. We report the average score of all 10000 mutations, the average of the top 1000, and the top 100 as determined by the oracle. Bold values are the highest average fitness value. *ICE* generates the highest quality mutations.

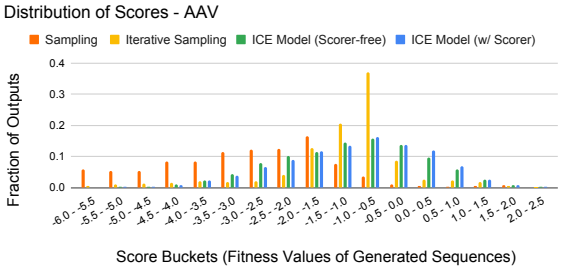


Figure 4: Histogram of fitness values of mutants generated by each approach on the AAV Task (higher scores are better). *ICE* generates outperforms *Sampling* and *Iterative Sampling*.

7.2. Results

ICE* model extrapolates better than *Iterative Sampling From Table 4, we see that *ICE* with *Scorer-Free* and *Scorer-Guided* inference achieves a higher success rate of extrapolation than *Sampling* and *Iterative Sampling* respectively. We also observe that *ICE* with *Scorer-Guided* inference achieves a higher average fitness score than the baselines on the total library of 10000 mutations as well as the subsets of Top-100 and Top-1000 mutations generated by each method. Lastly, it is also desirable to generate a library of mutations that not only achieves high fitness values but also exhibits diversity (Calcedo et al., 2009). We observe that *ICE* generates diverse and high-quality mutations by examining the edit distance between the mutations generated and the wild-type in Appendix C.1.

The scorer is less effective on AAV From Table 4, we see that the performance of both methods on the *training region* and *extrapolation region* targets when using the scorer improves only marginally over the scorer-free setups. The distribution of scores (Figure 4) also shows a similar trend. We see that, for both methods, the mode of the distribution of scores is within the *training region* itself, close to the boundary of the *extrapolation region* (Figure 4). The distribution for *ICE* is much flatter, which is why it achieves higher extrapolation success rates compared to *Iterative Sampling*. Since the generation process begins at the edge of the *training region* (zero), we expect the scorer to not offer much reliable guidance in AAV.

8. Conclusion

We presented *Iterative Controlled Extrapolation* (*ICE*), an iterative approach to extrapolative controlled generation. Our method considerably outperforms existing approaches to controllable generation and more complex extrapolative techniques on both NLP and protein design tasks. Potential future directions include extending the iterative approach to multiple attributes to generate sequences that compose them in novel ways, training scorers that generalize to the extrapolation region, and improving our synthetic data creation techniques by incorporating additional domain knowledge.

Acknowledgements

We thank David Belanger, Lucy Colwell, and Nitish Joshi for their valuable discussion and feedback during the course of the project. This work was undertaken as part of the Google Research Collabs program. This work is also supported by the Samsung Advanced Institute of Technology (Next Generation Deep Learning: From Pattern Recognition to AI), the National Science Foundation under Grant No. 1922658, and a gift from AWS AI.

References

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., and Mané, D. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

Angermueller, C., Belanger, D., Gane, A., Mariet, Z., Dohan, D., Murphy, K., Colwell, L., and Sculley, D. Population-based black-box optimization for biological sequence design. In *International Conference on Machine Learning*, pp. 324–334. PMLR, 2020a.

Angermueller, C., Dohan, D., Belanger, D., Deshpande, R., Murphy, K., and Colwell, L. Model-based reinforcement learning for biological sequence design. In *International Conference on Learning Representations*, 2020b. URL https://openreview.net/forum?id=HklxbgBKvr.

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

Bloom, J. D., Labthavikul, S. T., Otey, C. R., and Arnold, F. H. Protein stability promotes evolvability. *Proceedings of the National Academy of Sciences*, 103(15):5869–5874, 2006. doi: 10.1073/pnas.0510098103. URL https://www.pnas.org/doi/abs/10.1073/pnas.0510098103.

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.

Calcedo, R., Vandenberghe, L. H., Gao, G., Lin, J., and Wilson, J. M. Worldwide epidemiology of neutralizing antibodies to adeno-associated viruses. *The Journal of infectious diseases*, 199(3):381–390, 2009.

Chan, A., Madani, A., Krause, B., and Naik, N. Deep extrapolation for attribute-enhanced generation. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021a. URL https://openreview.net/forum?id=NCDMYD2y5kK.

Chan, A., Ong, Y.-S., Pung, B., Zhang, A., and Fu, J. Cocon: A self-supervised approach for controlled text generation. In *International Conference on Learning Representations*, 2021b. URL https://openreview.net/forum?id=VD_ozqvBy4W.

Chan, H. P., Wang, L., and King, I. Controllable summarization with constrained Markov decision process. *Transactions of the Association for Computational Linguistics*, 9:1213–1232, 2021c. doi: 10.1162/tacl_a_00423. URL https://aclanthology.org/2021.tacl-1.72.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=a7APmM4B9d.

Dallago, C., Mou, J., Johnston, K. E., Wittmann, B., Bhattacharya, N., Goldman, S., Madani, A., and Yang, K. K. FLIP: Benchmark tasks in fitness landscape inference for proteins. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021. URL https://openreview.net/forum?id=p2dMLEwL8tF.

Dathathri, S., Madotto, A., Lan, J., Hung, J., Frank, E., Molino, P., Yosinski, J., and Liu, R. Plug and play language models: A simple approach to controlled text generation. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1edEyBKDS.

Deller, M. C., Kong, L., and Rupp, B. Protein stability: a crystallographer's perspective. *Acta Crystallographica Section F: Structural Biology Communications*, 72(2): 72–95, 2016.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

Elnaggar, A., Heinzinger, M., Dallago, C., Rehawi, G., Wang, Y., Jones, L., Gibbs, T., Feher, T., Angerer, C., Steinegger, M., et al. Prottrans: Toward understanding the language of life through self-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 44(10):7112–7127, 2021.

Freschlin, C. R., Fahlberg, S. A., and Romero, P. A. Machine learning to navigate fitness landscapes for protein engineering. *Current Opinion in Biotechnology*, 75: 102713, 2022.

Gehman, S., Gururangan, S., Sap, M., Choi, Y., and Smith, N. A. RealToxicityPrompts: Evaluating neural toxic degeneration in language models. In *Findings of the Association for Computational Linguis-*

tics: EMNLP 2020, pp. 3356–3369, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.301. URL https://aclanthology.org/2020.findings-emnlp.301.

Gligorijević, V., Berenberg, D., Ra, S., Watkins, A., Kelow, S., Cho, K., and Bonneau, R. Function-guided protein design by deep manifold sampling. *bioRxiv*, 2021. doi: 10.1101/2021.12.22.473759. URL https://www.biorxiv.org/content/early/2021/12/23/2021.12.22.473759.

Gong, H., Bhat, S., Wu, L., Xiong, J., and Hwu, W.-M. Reinforcement learning based text style transfer without parallel training corpus. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 3168–3180, 2019.

Guu, K., Hashimoto, T. B., Oren, Y., and Liang, P. Generating sentences by editing prototypes. *Transactions of the Association for Computational Linguistics*, 6:437–450, 2018.

He, H., Peng, N., and Liang, P. Pun generation with surprise. In *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

Ibarz, B., Leike, J., Pohlen, T., Irving, G., Legg, S., and Amodei, D. Reward learning from human preferences and demonstrations in atari. *Advances in neural information processing systems*, 31, 2018.

Jain, A. and Berg-Kirkpatrick, T. An empirical study of extrapolation in text generation with scalar control. *arXiv preprint arXiv:2104.07910*, 2021.

Keskar, N. S., McCann, B., Varshney, L. R., Xiong, C., and Socher, R. CTRL: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*, 2019.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 7871–7880, 2020.

Li, X. L., Thickstun, J., Gulrajani, I., Liang, P., and Hashimoto, T. Diffusion-LM improves controllable text generation. In *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=3s9IrEsjLyk.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Lu, X., Welleck, S., Hessel, J., Jiang, L., Qin, L., West, P., Ammanabrolu, P., and Choi, Y. Quark: Controllable text generation with reinforced unlearning. *Advances in neural information processing systems*, 35:27591–27609, 2022.

Lyu, Y., Liang, P. P., Pham, H., Hovy, E., Póczos, B., Salakhutdinov, R., and Morency, L.-P. StylePTB: A compositional benchmark for fine-grained controllable text style transfer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2116–2138, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.171. URL https://aclanthology.org/2021.naacl-main.171.

Madani, A., McCann, B., Naik, N., Keskar, N. S., Anand, N., Eguchi, R. R., Huang, P.-S., and Socher, R. ProGen: Language modeling for protein generation. *arXiv preprint arXiv:2004.03497*, 2020.

Madani, A., Krause, B., Greene, E. R., Subramanian, S., Mohr, B. P., Holton, J. M., Olmos, J. L., Xiong, C., Sun, Z. Z., Socher, R., Fraser, J. S., and Naik, N. Deep neural language modeling enables functional protein generation across families. *bioRxiv*, 2021. doi: 10.1101/2021.07.18.452833. URL https://www.biorxiv.org/content/early/2021/07/18/2021.07.18.452833.

Madani, A., Krause, B., Greene, E. R., Subramanian, S., Mohr, B. P., Holton, J. M., Olmos Jr, J. L., Xiong, C., Sun, Z. Z., Socher, R., et al. Large language models generate functional protein sequences across diverse families. *Nature Biotechnology*, pp. 1–8, 2023.

Mallinson, J., Adamek, J., Malmi, E., and Severyn, A. EdiT5: Semi-autoregressive text editing with t5 warm-start. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 2126–2138, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.findings-emnlp.156.

Mueller, J., Gifford, D., and Jaakkola, T. Sequence to better sequence: continuous revision of combinatorial structures. In *International Conference on Machine Learning*, pp. 2536–2544. PMLR, 2017.

Novak, R., Auli, M., and Grangier, D. Iterative refinement for machine translation. *arXiv preprint arXiv:1610.06602*, 2016.

Pang, R. Y., Padmakumar, V., Sellam, T., Parikh, A. P., and He, H. Reward gaming in conditional text generation. *arXiv preprint arXiv:2211.08714*, 2022.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21 (1), June 2022. ISSN 1532-4435.

Ren, Z., Li, J., Ding, F., Zhou, Y., Ma, J., and Peng, J. Proximal exploration for model-guided protein sequence design. In *International Conference on Machine Learning*, pp. 18520–18536. PMLR, 2022.

Romero, P. A. and Arnold, F. H. Exploring protein fitness landscapes by directed evolution. *Nature Reviews Molecular Cell Biology*, 10(12):866–876, 2009.

Russell, S., Bennett, J., Wellman, J. A., Chung, D. C., Yu, Z.-F., Tillman, A., Wittes, J., Pappas, J., Elci, O., McCague, S., et al. Efficacy and safety of voretigene neparvovec (aav2-hrpe65v2) in patients with rpe65-mediated inherited retinal dystrophy: a randomised, controlled, open-label, phase 3 trial. *The Lancet*, 390(10097):849–860, 2017.

Schymkowitz, J., Borg, J., Stricher, F., Nys, R., Rousseau, F., and Serrano, L. The foldx web server: an online force field. *Nucleic Acids Research*, 33(suppl_2):W382–W388, 2005.

Shire, S. J., Shahrokh, Z., and Liu, J. Challenges in the development of high protein concentration formulations. *Journal of pharmaceutical sciences*, 93(6):1390–1402, 2004.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

Verkuil, R., Kabeli, O., Du, Y., Wicky, B. I., Milles, L. F., Dauparas, J., Baker, D., Ovchinnikov, S., Sercu, T., and Rives, A. Language models generalize beyond natural proteins. *bioRxiv*, 2022.

Wang, W. Instability, stabilization, and formulation of liquid protein pharmaceuticals. *International journal of pharmaceutics*, 185(2):129–188, 1999.

Webber, M. J., Appel, E. A., Vinciguerra, B., Cortinas, A. B., Thapa, L. S., Jhunjhunwala, S., Isaacs, L., Langer, R., and Anderson, D. G. Supramolecular pegylation of biopharmaceuticals. *Proceedings of the National Academy of Sciences*, 113(50):14189–14194, 2016.

Welleck, S., Lu, X., West, P., Brahman, F., Shen, T., Khashabi, D., and Choi, Y. Generating sequences by learning to self-correct. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=hH36JeQZDaO.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL https://aclanthology.org/2020.emnlp-demos.6.

Yang, K. and Klein, D. FUDGE: Controlled text generation with future discriminators. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3511–3535, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.276. URL https://aclanthology.org/2021.naacl-main.276.

Yang, K. K., Wu, Z., and Arnold, F. H. Machine-learning-guided directed evolution for protein engineering. *Nature Methods*, 16(8):687–694, 2019.

Zhang, X., Zhao, J., and LeCun, Y. Character-level convolutional networks for text classification. *Advances in Neural Information Processing Systems*, 28, 2015.

A. Limitations

Creation of synthetic data can introduce hallucinations in natural language Our method relies on masked language modeling to create minimally perturbed pairs of sequences (Section 3.2). In natural language tasks, this can result in a perturbed sequence that is slightly different in meaning from the source sequence. As a result, the *ICE* model when trained can also alter the meaning of the sequence. In particular, we want to note that certain kinds of hallucinations from text generation models can be harmful if used without proper consideration. Specifically, in Table 7, it is acceptable for the model to edit the sentiment associated with the food or ambiance at the restaurant but we want the model to retain the basic information that the writer and his partner are eating at a sushi restaurant in Scottsdale. Going forward, we intend to investigate better strategies for synthetic data creation to measure and mitigate this occurrence.

Assumption that edits in the *training region* generalize to *extrapolation region* Our work relies on training a model on perturbations made on sequences belonging to the *training region*. We then repeatedly make edits to increase or decrease the score into the *extrapolation region*. While our experiments show promising results, we believe that this assumption does not equally hold for all tasks and domains. We intend to study this further going forward.

Relying on trained models to score sequences For evaluation of the sentiment control and the AAV tasks, we train classifier models to measure the attribute values of the sequences. These models only estimate the ground truth attribute values and can end up learning spurious correlations from the datasets. We note that these are to be used as a means to benchmark our method against the various baselines. Particularly in the case of proteins such as AAV, prior to any real-world usage, a detailed analysis of the oracle models or real-life wet lab experiments should be performed.

Inference for iterative methods is slow By the nature of our method, iteratively editing a sequence is much slower in terms of inference time as compared to a single-step edit by a model such as *Genhance*.

B. Additional Model Training Details

We fine-tune all of the language models for our experiments using the HuggingFace library (Wolf et al., 2020). All of the code used for our experiments and trained models is available at `https://github.com/vishakhpk/iter-extrapolation`.

Sentiment Control The scorer and oracle model used for evaluation are fine-tuned RoBERTa-Large (Liu et al., 2019) models. The oracle is trained on the entire Yelp dataset. The scorer is trained on those examples with a sentiment from 2 to 4. Both the scorer and oracle are fine-tuned to optimize the mean-squared error loss on the gold labels from the dataset. We create paired data to train the *ICE* generator model using the scorer and a pre-trained T5-Base (Raffel et al., 2022) model. We create 100K pairs and fine-tune T5-Base to serve as the *ICE* generator. The hyperparameter $\delta = 0.4$ used to filter synthetic pairs was selected based on a small internal pilot. We fine-tune T5-Base to generate the output of the synthetic pairs given the input sequences optimizing the cross-entropy loss on the output tokens. For each of these, we use the recommended hyperparameters from the HuggingFace repository and sweep learning rates from $1e{-}6$ to $1e{-}3$.

ACE2 For ACE2, we fine-tune a ProtBert (Elnaggar et al., 2021) model, made available via the HuggingFace, to predict the *ddG* values given the mutants from the dataset released by Chan et al. (2021a). Here we optimize the mean-squared error loss on the gold labels, selecting the optimum checkpoint using the validation loss. We use this to create a synthetic dataset of 1M pairs which is used to fine-tune the *ICE* generator model. We fine-tune Prot-T5-XL (Elnaggar et al., 2021) on these pairs to generate the output of the synthetic pairs given the input sequences optimizing the cross-entropy loss on the output tokens. We again use the recommended hyperparameters from the HuggingFace repository and sweep learning rates from $1e{-}6$ to $1e{-}3$. For scoring with FoldX, we match the parameters from (Chan et al., 2021a).

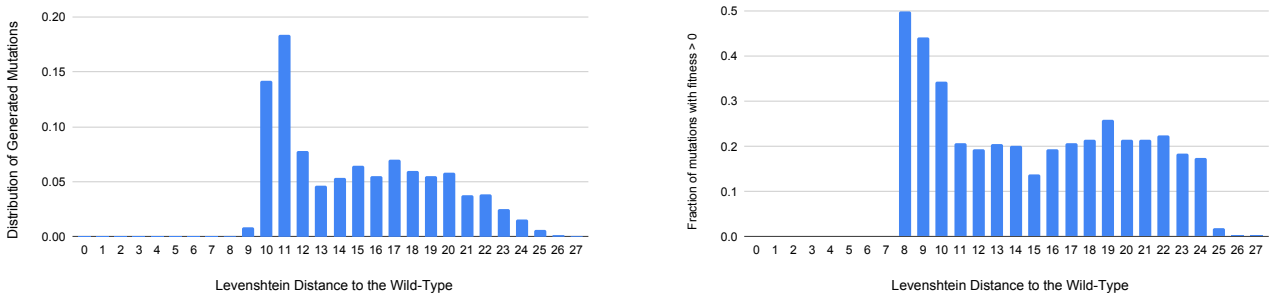
AAV The scorer and oracle models for the AAV task are CNN models that accept the protein sequence as a string and output a real number corresponding to the fitness value. We select the model architecture according to the parameters specified in the FLIP benchmark (Dallago et al., 2021). We follow the same as the obtained the highest test spearman correlation for the AAV *low-vs-high* split. Both CNN models are trained from the repository of the benchmark optimizing the mean squared-error loss on the fitness values. We use the scorer to create 1M synthetic pairs to train the *ICE* generator model optimizing the cross-entropy loss of the output tokens given the input protein sequence and corresponding control tag.

C. Additional Findings

C.1. Exploring Diversity in AAV Mutants

While AAV capsids hold promise for gene therapy, the immunity from prior AAV exposure excludes 20–80% of the population from such treatments (Calcedo et al., 2009). Thus, it is essential to not only generate AAV mutants of high fitness, but also of significant diversity from the wild type. To this end, in Figure 4, we analyze the distribution of sequences generated by our model (in the 10th iteration) as a function of their Levenshtein distance from the wild-type. We see that while the majority of mutations generated have an edit distance of around 8–10, the model generates mutations having as far as 25 edits from the wild-type (Figure 5a). However, we see that even when the model makes over 20 edits, the fraction of examples within this bucket is still 0.2, showing a large diversity in the mutations generated (Figure 5b).

We note that the model generates a mutant at a diverse range of levenstein distances from the wild type (8 to 27). Moreover, *ICE* displays strong performance throughout this range according to our oracle (Figure 5b), demonstrating its potential to generate both viable and diverse mutants of AAV.



(a) (b)

Figure 5: We plot the fraction of sequences for a given Levenshtein distance away from the wild type (Figure 5a). Figure 5b shows the fraction of generated sequences that are better than the wild type (according to the oracle) as a function of the Levenshtein distance, showing the potential of *ICE* to generate both diverse and viable mutants.

C.2. Additional Results on Sentiment Control

Table 7 shows an example of the editing process, increasing the sentiment score of the input review iteratively. In addition to the results from Table 1, we report a few variants of *ICE* and *Genhance*. For *ICE*, the masking strategy to create synthetic paired data involves sampling a location in the sequence to start the mask using a Bernoulli distribution ($p = 0.8$) and then selecting the length of the mask (in terms of tokens masked) by sampling from a truncated Poisson distribution. The results presented in Table 1 correspond to the *Super Large* variant in Table 6 where $\lambda = 6$ and the maximum span size is set to 12. We also report three other variants of the masking strategy *Small* ($\lambda = 3$, maximum of 6), *Medium* ($\lambda = 4$, maximum of 8) and *Large* ($\lambda = 5$, maximum of 10). We observed the best extrapolation results on the *Super Large* variant and used this masking strategy to report the *Sampling* and *Iterative Sampling* baselines. We also report two variants of *Genhance* where we vary the total number of output sequences generated for each example. As we increase n, the model predictably performs better at extrapolation but we see that the directly comparable variant, $n = 50$, is outperformed by *ICE*.

C.3. Sensitivity to Hyperparameters of Generation

To study the interaction between the generation hyperparameters and the number of iterations at inference time, we ran both scorer-free inference varying the beam size and scorer-guided inference varying k in top-k for the *ACE2* task. In all cases, we generated 1000 mutations. We present the results at iteration 2,5, and 10 in Table 8. Each cell of the table represents the fraction of mutations with *ddG* value lower than the corresponding target rounded off to three decimal places. The rows corresponding to top-$k = 5$ and beam size 5 at iteration 10 were included in Table 2.

Overall, we find that the results at the end of the inference process (iteration 10) are largely stable w.r.t. these hyperparameters. In particular, when increasing k for top-k sampling, we see a slight drop in performance, which might be due to the small vocabulary size of protein sequences (a total of 20). Similarly, for scorer-free inference, as we decrease beam size to 3 we

Target Sentiment Score	Training Region			Extrapolation Region		
	3.5	2.5	Average	4.5	1.5	Average
Score-Conditioned Baseline	0.780	0.766	0.773	0.212	0.217	0.215
PPLM	0.534	0.516	0.522	0.081	0.065	0.077
Sampling	0.362	0.259	0.310	0.061	0.050	0.056
Iterative Sampling	0.668	0.657	0.663	0.320	0.328	0.324
Genhance (n = 1)	0.407	0.167	0.287	0.063	0.025	0.044
Genhance (n = 50)	0.982	0.833	0.908	0.482	0.291	0.387
Genhance (n = 100)	**0.995**	0.912	**0.954**	**0.670**	0.429	0.550
ICE w/ Scorer – Small	0.962	0.98	0.971	0.514	0.344	0.429
Medium	0.945	0.870	0.908	0.636	0.499	0.567
Large	0.953	0.884	0.918	0.649	0.555	0.602
Super Large	0.943	0.900	0.921	0.638	**0.582**	**0.610**
ICE Scorer-Free	0.976	**0.918**	0.947	0.446	0.305	0.376

Table 6: Results on sentiment control in both the training and extrapolation regions including ablations of our model and Genhance. Evaluation is done by measuring the fraction of examples that have a sentiment value greater than (or less than) a target score as determined by the oracle scorer. Bold values are the highest success rates for each target. *ICE* achieves the highest rate of extrapolation.

obtain slightly better performance in the training region with a small drop-off for extrapolation. Increasing the beam size to 10 mildly decreases performance.

We find that the iteration number is a reliable indicator of the extrapolation performance with little change in performance observed due to the top-k and beam size hyperparameters (within each specific iteration). At iteration 2, when guided by the scorer, a higher top-k value results in better performance as the model samples more diverse generations, and the scorer can reliably select good sequences to obtain better performance on targets in the training region. Similarly, for scorer-free inference, a higher beam size also improves performance on the targets in the training region. However as we increase the number of iterations to iteration 5 and 10, this effect largely evens out.

C.4. Stopping Criteria

Reliably identifying when the generation model has reached a target score is difficult due to the extrapolative nature of the task. Specifically, if we had a way to know when the generator model has achieved a target score in the extrapolation region, then this supervision could directly be used to train the generator itself. One option is to use the scorer, f_s. However, we observed the output of f_s plateau near the boundary of the training region, limiting its reliability as a stopping condition in the extrapolation region. To illustrate this, we present the average output score in the *ACE2* task as a function of 10 iterations in Table 9. We observed that the output score remained largely constant beyond iteration 7. Hence we settled on setting the number of iterations to greater than the plateau point of the scorer, such as 10, and found that this worked well across our 3 datasets without further tuning. However, we acknowledge the need for a more principled stopping condition as an open problem in this setting.

Iteration	Text
1	<inc>This place is just okay. The wait is way too long, the counter person is irritating, and the unsupervised children are disruptive. The fill and nail art are decent, but not worth the wait. Lisa was fast, and I dried quickly. I found the nail art I wanted on Pinterest, and she did a decent job replicating it. She also charged my phone, which was thoughtful. If you're going, try to get there early, as it gets crowded later in the day. It's not amazing, but it's worth checking out.
2	<inc>This place is average at best. The wait is too long, the counter person is annoying, and the unsupervised children are a problem. The fill and nail art are decent, but nothing special. Lisa was quick, and I dried quickly. I found the nail art I wanted on Pinterest, and she did a good job replicating it. She also charged my phone, which was nice. If you're going, try to get there early, as it gets busier as the day goes on. It's worth a visit, but don't expect anything amazing.
3	<inc>This place is decent, but has some flaws. The wait is long, the counter person is too pushy, and the unsupervised children are distracting. The fill and nail art are really good, though, and Lisa was super quick. I found the nail art I wanted on Pinterest, and she replicated it perfectly. Bonus points for charging my phone. If you're thinking about going, try to get there early, as the line gets longer later in the day. Despite the flaws, it's still worth a visit.
4	<inc>This place is good, but not great. The wait is a bit long, the counter person is pushy, and the unsupervised children are annoying. However, the fill and nail art are fantastic, so it's worth it. Lisa was quick, and I was drying in no time. I found the nail art I wanted on Pinterest, and she copied it perfectly. She even charged my phone, which was nice. If you plan to visit, go early as it gets busier closer to closing time.
5	<inc>This place is pretty good! The wait, pushy counter person, and unsupervised children are easy to ignore when I leave with the best fill and nail art ever! Lisa was lightning-fast, and I was drying in under 15 minutes! I found the nail art I wanted on Pinterest, and she copied it flawlessly! As a bonus, she charged my phone, which was a nice gesture! If you're planning on going, try to arrive early, as the line gets longer closer to closing. But trust me, it's worth the wait!
6	<inc>This place is great! The wait, pushy counter person, and unsupervised children are an easy overlook when I finally leave with the best fill and nail art I've ever had! Lisa was super quick, had me drying in less than 15 minutes of sitting down in her chair! I found the nail art I wanted (she copied it perfectly, by the way) on pintrest, but just as I sat down, my phone died. She pulled out her charger, and charged my phone! Where else has anyone done this? Nowhere. Just a heads up, go early, if you can, as it gets closer to close, more and more people line up. :) it's so worth the wait, though!!

Table 7: Trajectory of improving the sentiment associated with a review using *ICE*.

Target *ddG* Value			Training Region -1	Training Region -2.5	Extrapolation Region -5	Extrapolation Region -6	Extrapolation Region -7
ICE w/ Scorer: Varying K for sampling	Iteration = 10	TopK = 15	0.997	0.964	0.249	0.083	0.01
		TopK = 10	0.998	0.966	0.283	0.091	0.016
		TopK = 5	**0.998**	**0.974**	**0.362**	**0.098**	**0.019**
	Iteration = 5	TopK = 15	0.982	0.648	0.041	0.004	0.000
		TopK = 10	0.981	0.646	0.040	0.004	0.000
		TopK = 5	0.978	0.647	0.042	0.005	0.001
	Iteration = 2	TopK = 15	0.711	0.093	0.002	0.000	0.000
		TopK = 10	0.703	0.090	0.001	0.000	0.000
		TopK = 5	0.674	0.086	0.001	0.000	0.000
ICE Scorer-Free: Varying beam size	Iteration = 10	Beam Size = 10	0.930	0.572	0.059	0.013	0.000
		Beam Size = 5	0.945	0.598	**0.062**	**0.017**	**0.002**
		Beam Size = 3	**0.959**	**0.623**	0.060	0.016	0.000
	Iteration = 5	Beam Size = 10	0.852	0.440	0.030	0.006	0.000
		Beam Size = 5	0.847	0.437	0.026	0.005	0.000
		Beam Size = 3	0.844	0.419	0.023	0.004	0.000
	Iteration = 2	Beam Size = 10	0.620	0.182	0.001	0.000	0.000
		Beam Size = 5	0.567	0.155	0.001	0.000	0.000
		Beam Size = 3	0.526	0.143	0.000	0.000	0.000

Table 8: Evaluation on the *ACE2* task to study the interaction between the generation hyperparameters and the number of iterations at inference time. Each table cell represents the fraction of mutations with a *ddG* value lower than the corresponding target. We vary k for top-k sampling for scorer-guided inference and vary beam size for scorer-free inference. We find that the results are largely stable with respect to these hyperparameters at the end of inference (i.e., iteration 10). Early on during inference (i.e., iteration 2), we find that a higher top-k value and beam size respectively result in better performance but this largely evens out by iteration 5 and 10.

Iteration	Average Score
1	-0.673
2	-2.051
3	-2.879
4	-3.272
5	-3.446
6	-3.522
7	-3.551
8	-3.558
9	-3.555
10	-3.567

Table 9: Average output scores of f_s as a function of iterations in the *ACE2* task. Each cell is an average of the scores assigned to the $10,000$ mutants generated with scorer-guided inference in Table 2. We observe that the output of f_s plateaus near the boundary of the training region at around -3.5 making it unreliable as a stopping condition for the generation process.