PandoGen: Generating complete instances of future SARS-CoV2 sequences using Deep Learning

Anand Ramachandran[†], Steven S. Lumetta, Deming Chen
University of Illinois at Urbana-Champaign

Abstract

Deep generative models have achieved state-of-the-art performance in many areas including image generation, code generation and natural language generation. We explore the use of deep generative models in producing complete instances of as-yet undiscovered SARS-CoV2 Spike protein sequences. The Spike protein is the primary initiator of infection by the SARS-CoV2 virus, and hence, the ability to predict future manifestations of the Spike protein is invaluable, enabling critical tasks such as advance validation of pharmaceutical interventions. We examine specific requirements of generating sequences for a pandemic and formulate a novel framework for training models for these requirements. Our solution only uses sequence information submitted in SARS-CoV2 repositories without the need for additional laboratory experiments. Resulting models substantially outperform a state-of-the-art generative model for protein sequences finetuned on SARS-CoV2 data. Samples produced from our models are four times as likely to be novel and real SARS-CoV2, and ten times as infectious, cumulatively. We find that among higher ranked sequences generated from our model, over 70% are discovered in the future, over twice the rate of the baseline. Our models represent a promising source of hypothetical SARS-CoV2 sequences, thus providing a key tool for advance preparation against the pandemic. PandoGen is available at https://github.com/UIUC-ChenLab/PandoGen

1 Introduction

Deep generative models have achieved breakthrough results in generating complete, self-contained, and realistic examples of complex entities such as images [RPG+21], language pieces [Ope23], and computer code [LCC+22]. Especially, a class of models called Large Language Models (LLMs) have recently achieved ground-breaking performances in following human instructions faithfully [Ope23] [OWJ+22] and performing at the human-level in competitive programming [LCC+22]. Models based on LLM architectures have recently gained popularity in modeling biological sequences as well [RMS+19] [EHD+21] and can be useful tools in generating samples with interesting properties [FSH22] [SRK+21]. When applied to proteins these models are referred to as Protein Language Models (PLMs) in literature.

In this article, we address a hitherto unaddressed question: how can a PLM be trained to generate complete, self-contained, realistic, and unknown instances of biological sequences in a pandemic situation, which may appear in the future? We try to answer this question for the SARS-CoV2 virus first discovered in December 2019, and has since unleashed a pandemic that has resulted in massive loss of life, and vast economic damage. Among the SARS-CoV2 proteins, the Spike protein holds special significance. The Spike protein binds to the ACE-2 enzyme found on the cell surface of the host, gaining entrance to the cell. The efficiency with which this procedure is carried out reflects the infectiousness of the virus [Xia21]. As a result, many treatment methods such as antibodies target the Spike protein for neutralization. Advance knowledge of future Spike protein sequences can hence provide an important resource for stress-testing interventions such as vaccines and antibody treatments. Hence we focus our efforts on the Spike protein.

Deep Mutational Scanning (DMS) [FF14] has become popular as a method to probe properties of protein sequences, and offers one avenue for performing tests of hypothetical Spike protein sequences. Recently, a DMS method was developed that could probe the entirety of the Spike protein sequence [DCR+23], capable of scanning sequence sets numbering to the order of 10^5. Through experiments such as these, a Spike protein variant's ability to infiltrate the cell through receptor binding, as well as its ability to escape or withstand antibody binding and neutralization can be studied. However, given

[†]Work does not relate to author's position at Amazon

that the entire length of the Spike protein sequence has seen mutations (the Spike sequence is longer than 1200 amino acids for reference), simply the space of combinations of all known mutations of the Spike protein is astronomical. Hence, the problem of which hypothetical Spike sequences to study through these experiments becomes salient, given that budget of sequences that can be studied is not unlimited. Deep generative models, in principle, can bridge the gap by providing plausible sequences to test.

However, in practice, it is not clear how the models can be trained to achieve this. Deep generative models are typically used to parameterize the probability distribution of the targeted data (e.g., sequences). Traditionally, the models are trained using a known set of sequences in the domain of interest. The models learn the distribution of the training data, by maximizing the likelihood of the training examples, and allow sampling from that distribution. In a pandemic situation, we are interested in as yet undiscovered sequences which will potentially be found in the future. There are distinct characteristics between sequences known until a point of time, and new sequences to be discovered in the future. For example, as the pandemic progresses, sequences accumulate more mutations. So, future sequences likely carry a larger number of mutations than the sequences on which models are trained. Sampling from the training distribution, which is based on sequences known until a specific point of time, is not a good way to model such *arrow of time* effects. A model may have learnt the characteristics of the sequences, but the standard training method does not include a way to train the models to project these characteristics into the future, away from a known set of sequences.

Also, model predictions must be *salient*. In the context of SARS-CoV2, this means that sampled sequences must have non-trivial case counts. For instance, out of all Spike protein sequences submitted to GISAID [SM17] as of 11/29/2022, there are 762,192 unique sequences. Out of these, only 47,839 sequences (or 6.2%) had been reported 10 or more times in the database and 523,582 (or 69%) had been reported exactly once. Given a large number of non-salient sequences among the known set of sequences, the models need to know that some sequences are more important than others, but at the same time, that all sequences carry important information about plausible structure and function related to the organism. An obvious way to include this information in training is to have the training set reflect the true distribution of the underlying data, by repeating sequences in proportion to their GISAID case counts. So, a sequence having n case counts in GISAID would be repeated n times in the training dataset. This would teach the model which sequences are more salient, and what are their characteristics. However, this does not answer the question of how the distributional characteristics found within existing sequences can be extrapolated to produce novel sequences that haven't occurred yet.

Due to these limitations of standard likelihood maximization methods, we may need to look to alternatives. There exist multiple strategies to train generative models through guidance from a second, discriminative, model, which rates the output of the generative model. In Generative Adversarial Networks (GAN) [GPAM+20], the discriminative model learns to predict whether a sequence presented to it is sampled from a set of known, real sequences, or produced from the generative model. During training, the generative model's goal is to learn to fool the discriminative model into thinking that data generated from the generative model is real. In Natural Language Processing (NLP), a method called Reinforcement Learning Through Human Feedback (RLHF) is becoming popular as a means to improving interactions between humans and LLMs. Here, a discriminative model is trained to learn how humans would score different generative model responses to human queries [OWJ+22] [Ope23]. This is accomplished via obtaining a human-labeled dataset on which the discriminative model is trained, thereby infusing external information into the training process. The generative model then attempts to improve itself trying to get better "rewards" from the discriminative model. Neither approach fits our requirement directly. The case of GANs is not a good fit for the pandemic scenario, as it cannot work with the arrow of time problem discussed above. Since the set of real sequences used in GAN training is limited to known sequences as of today, it is not clear how to train the discriminative model to learn to label future pandemic sequences correctly, such as ones that come from a generative model which is trying to predict the future. In addition, the method doesn't address the sequence salience issue discussed above. RLHF in NLP is used for improving model-generated continuations of a human provided prompt or query. For example, a human may ask a question regarding a certain topic, and the goal of RLHF training is to improve the response from the model given the human question as prefix. In our case, there is no such prompt - we ask the model to simply generate sequences without providing any sequence as a prefix. Furthermore, in RLHF, the discriminative model is trained using human

labels applied to textual data indicating the desirability of the data as model outputs for the given human inputs. In our case of protein sequences, human annotations do not work, and it is unclear what to replace such annotation with, so that labeled data can be obtained for training the discriminative model. In addition, RLHF doesn't address the arrow of time issues discussed previously, where models learn from the training distribution, but produce sequences mirroring future data distribution.

To formulate a solution to these problems, we look at sequence data already available at a given point of time in the pandemic, along with submission dates and location data. The first novel piece of the solution is to device a discriminator training method that uses this available information, without needing expensive external knowledge incorporation steps such as annotations or lab experiments. Viewing the spread of the disease as games played among the different Spike sequences, we devise a method to train a discriminative model that learns the probability distribution of the outcome in pairwise competitions among Spike sequences. The competitions are setup so that sequences with higher case counts in GISAID have a higher chance to win. However, a sequence discovered under one set of circumstances (e.g., available sequencing capacity, disease penetration, treatments and other preventative measures) cannot be fairly compared with a sequence discovered under a different set of circumstances. We use this guiding principle to ensure that only fair games are played. During this training, we expect the model to learn a notion of Spike protein success from Spike protein sequence, scoring more successful sequences higher. The next novel piece of the solution is to teach the generative model patterns related to successful sequences, and to apply this knowledge to the generation of new sequences, rather than sequences from the training distribution. Specifically, we use the discriminative model trained as described above to guide the finetuning of a likelihood maximized generative model where (1) the generative model learns to generate successful sequences and to avoid less successful sequences and (2) the generative model preserves sequence understanding it learnt from the training set, but learns to avoid generating sequences from the training set. Together these steps guide the generative model towards learning sequence characteristics of successful sequences, and apply them to novel, unknown sequences, potentially solving both of the requirements for the problem.

In detail, we develop a four-stage training pipeline using a class of models called deep autoregressive models, which have become the leading models for many types of sequential data applications [RNS+18] [RWC+19] [ODZ+16]. Deep autoregressive models implement sequence generation through iterative generation of the next sequence element. Hence the likelihood of a complete sequence may be written as follows.

$$P(X_{1:L}) = \Pi_{i=1}^{L} P(x_i \mid X_{1:i-1}; \Phi) \tag{1}$$

Here, the probability of a full sequence, $X_{1:L}$, is written as the product of the probability of each sequence element conditioned on the past sequence elements, $P(x_i \mid X_{1:i-1}; \Phi)$, where Φ represents the model's parameters which ultimately determine the probabilities. While the modeling scheme is a simple iteration, the underlying neural network can have hundreds of millions or billions of parameters, resulting in highly capable modeling. When training large neural networks, it is standard practice to pre-train them on unlabeled data in the domain. Such data is cheap to obtain due to lack of expensive processing steps such as labeling. For example, in NLP, data from the internet [RSR+20] [DSM+21] is used to pre-train large models. This pre-training step allows models to gain an understanding of the structure of the data in the domain of interest. Following this approach, the first stage in our pipeline is hence to train a deep autoregressive model on the UniProt dataset [uni17], specifically the UniRef50 sequence set, which contains tens of millions of protein sequences that have been reported so far. The training proceeds by maximizing the likelihood of the training dataset under the model, which can be summarized as follows [DEKM98], where $\mathcal{D}$ is the training dataset, and Φ^* represents model parameters after training.

$$\Phi^* = \arg\max_{\Phi} \log P(\mathcal{D} \mid \Phi) = \arg\max_{\Phi} \sum_{X^{(j)} \in \mathcal{D}} \sum_{i} \log P(x_i \mid X_{1:i-1}^{(j)}; \Phi) \tag{2}$$

In practice, for deep neural networks, this is implemented through mini-batch gradient descent. To pretrain our models, we used a version of the UniRef50 dataset that was available prior to December 2019 to ensure that the models do not see any SARS-CoV2 sequences during this stage.

The second stage is to finetune the model on SARS-CoV2 Spike protein sequences from GISAID, producing a SARS-CoV2 deep autoregressive (SDA) model. We select a date cut-off point and only sequences reported on or before this date were used for this finetuning step. This step also proceeds by

maximizing the likelihood of the training set, which in this case is the SARS-CoV2 training sequence set. This method, called Transfer Learning [RSR+20], is the canonical approach to finetune large models for a sub-domain (here, SARS-CoV2) by transferring knowledge from a large model trained on domain-level data (UniProt sequences).

There exist similar works in literature for stages one and two. For instance, Prot GPT2 [FSH22] and is a large protein generative models trained on the publicly available protein corpuses using likelihood maximization techniques. The model is derived from the popular GPT2 Large Language Model (LLM) that is popular in NLP [RWC+19], and is a deep autoregressive model similar to our case. The method has also been fine-tuned on Spike protein sequences similar to the second stage outlined above via Transfer Learning [Dho23]. The model was then asked to generate continuations of a specific localized sequence context within the Spike protein. The generated results were analyzed for individual mutations in the context of interest. However, the work does not examine the question of *de novo* generation of complete Spike protein sequences, where (1) the entire Spike protein rather than a local context is to be generated and (2) where not single mutations but combinations of mutations also need to valid, with potential for manifesting in nature, even though, Prot GPT2 is capable of generating complete instances of the Spike protein. In this article, we compare our results to that from Prot GPT2 operated as a *de novo* sequence generator.

The third stage in our training pipeline is to train a reward model to predict the potential success or infectiousness of a Spike sequence, as outlined above, by setting up *fair*, pairwise competitions among SARS-CoV2 sequences in the training period, and using a model to predict the winner. This stage uses the same training sequences as the SDA model, along with the geographical location of sequence reporting, and the date of reporting. Training uses only information from the training period. So if a sequence was discovered in the training period, only the instances of the sequence reported within the training period are used, and if the sequence continues to be reported in the future, that information is not used.

The fourth stage is to finetune the SDA model directly in its operational setting (i.e., sequence generation), guided by the reward model. An iterative procedure is used here, where the SDA generates sequences, each generated sequence is marked as known (based on what is known in the training period) or unknown, the reward model is run on each generated sequence to mark its infectiousness potential, and this annotated data is used to finetune the SDA model. The goal is to improve the SDA model's ability to generate novel sequences with high rewards. Note that no new external training data is used in the fourth-stage, and the generative model is guided by reward model during the training process. The training scheme is shown in Figure 1.

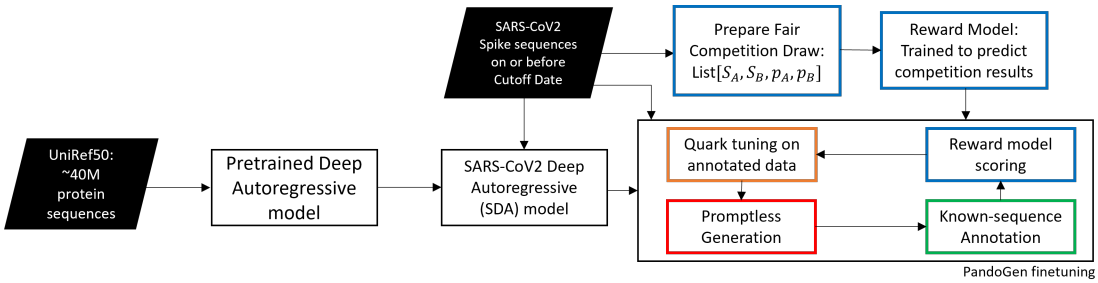

Figure 1: The PandoGen high-level training flow. The training flow starts with pre-training on the UniProt database, followed by finetuning on the SARS-CoV2 sequences to produce an SDA model. We then create a competition draw among known sequences based on the fair-competition principle described in Section 1 and train a reward model to learn the probability distribution of the outcome for games between two sequences. The SDA model is then finetuned in its operational setting through an iterative procedure involving, promptless generation, historical data look-up, reward model scoring, and tuning using the generated annotated, data. To perform model updates during training, we use a method based on Quark [LWJ+22]. The iterative process is terminated when reward improvements saturate.

As a result of this finetuning, we find that the resultant PandoGen model produces sequence samples with higher quality. The samples have higher fraction of real sequences among novel sequences, higher number of novel sequences per sample, and higher cumulative case counts than the baseline. The

model also has a wider operating range, meaning the samples generated from the model are more different from the training sequences than those generated from the baseline method. Overall, our method produces a model that is many times as effective as the baseline method, while being many times smaller.

2 Results

We designate sequences reported on or before 06/15/2021 as training sequences. Hence all models are trained using only sequences reported until that date. Model generation is benchmarked against sequences reported after this date.

As multiple cases may be reported for a given Spike protein sequence in GISAID in the training period, we distinguish between a training set which contains each sequence only once ("unique sequences") and a training set which contains each sequence as many times as it is reported in the training period in GISAID ("all sequences"). We train four models (1) the SDA model trained on unique sequences in the training period (2) Prot GPT2 finetuned on unique sequences in the training period ("Prot GPT2 unenumerated" model) (3) Prot GPT2 finetuned on all sequences in the training period ("Prot GPT2 enumerated" model) and (4) PandoGen. The Prot GPT2 models are selected as baselines as Prot-GPT2 is a state-of-the-art PLM, and this method also represents the traditional method for finetuning PLMs for a specific task. The SDA model is also added for comparison to show that any gains made by PandoGen are not due to differences in network architecture or other characteristics of the model compared to Prot GPT2, but due to the PandoGen training flow. The Prot-GPT2 models have close to 800 million parameters compared to approximately 200 million parameters for SDA and PandoGen. To compare inference performance, each model and sampling configuration is run six independent times, each time generating 2048 total samples.

2.1 Sampling algorithms and operating points

The unrestricted sampling process from a deep autoregressive model proceeds as follows

$$\hat{x}_i \sim P(x_i \mid X_{1:i-1}; \Phi) \tag{3}$$

Here, Φ represents the model parameters. As shown, the i-th sequence element, x_i, is sampled from the model distribution, which is conditioned on all previous sequence elements sampled from the model. This is iteratively performed until the end of sequence is encountered. This sampling process yields sequences with the maximum amount of output variation from the trained model. On the other extreme, the following greedy search algorithm is also available, which always yields exactly one sequence, representing a minimal amount of variation in the model output.

$$\hat{x}_i = \arg\max_{x_i} P(x_i \mid X_{1:i-1}; \Phi) \tag{4}$$

There are different methods to tune the model's output so that it lies somewhere between the two extremes. We adopt nucleus sampling [HBD+19], a widely popular sampling method used for LLMs.

To explain nucleus sampling, we define a few terms for clarity first. A sequence is an ordered tuple of sequence elements. Each sequence element is a token that is sampled from the set of all possible tokens that can appear in the sequence, called the sequence alphabet. Nucleus sampling filters away noisy cases from the sampling process in Equation 3 by defining a nucleus, which is a subset of the sequence alphabet, and then sampling from within that nucleus at each step. The nucleus at step i is defined as follows.

$$N_i = \arg\min_{||M||} \Big(\sum_{x \in M} P(x \mid X_{1:i-1}; \Phi) >= p \Big) \tag{5}$$

Here, M is a subset of the sequence alphabet. The nucleus, N_i, is the smallest subset of this set for which the sum of the probabilities at timestep i in the sequence generation process is at least p, a parameter of nucleus sampling. The tokens in the sequence alphabet outside N_i are discarded, and the ith sequence element is sampled from within N_i (the probabilities within N_i are renormalized for this purpose). The maximum value of p is 1 where the nucleus is the complete sequence alphabet. As

p decreases, fewer elements are kept inside the nucleus, reducing the number of potential outcomes for each sampling step in the sequence generation process. Essentially, this removes lower likelihood, potentially erroneous, elements from consideration at each sampling step. While in most cases, the tokens outside the nucleus are erroneous cases, when p is sufficiently small, the nucleus will begin to exclude non-erroneous high-information cases. So, at sufficiently low values of p, the model would fail to output interesting new information. Hence, varying p allows us to explore the trade-off between error control, and novelty of information in the sampled output.

While p gives us a knob to tune the sampling operation, the value of p alone does not convey a meaningful characterization of the operating point of the model in terms of characteristics of its sequence sample. To provide this characterization, we look to quantify the difference between a generated sample and the known sequences of SARS-CoV2. A higher degree of difference indicates that generated sequences are likely further apart, evolutionarily, from known sequences. It is natural to expect the models to have a higher error rate when the sample difference with respect to the training set is higher as models would need to predict further into the future in this case. We expect that lower values of p cause the sample difference to be lower and higher values allow generated sequences to deviate further from the training set.

To quantify sample difference from the known SARS-CoV2 corpus without expensive computational overheads, we formulate a method using k-mer matching. This is based on methods used to determine the quality of data in genome sequencing [HMRC21]. First, we collect all unique SARS-CoV2 spike sequences in the training period. We obtain all the k-mers in these sequences into a reference k-mer set ($k = 11$ in our experiments). Next, for each unique sequence in a generated sample, we count the number of k-mers in the sequence that are not found in the reference set. Using this, we find the average per-sequence k-mer novelty of the sample with the reference k-mer set. We designate this quantity as the measure of sample difference with respect to the reference (training) sample. Specifically, if S_i is the number of k-mers in the ith sequence that are not found in the reference k-mer set, the k-mer difference for the sample is $\frac{\sum_{i=1}^{N} S_i}{N}$, where N is the number of unique sequences in the sample. Naturally, the higher the k-mer difference of a sample, the further it is from the known set of SARS-CoV2 sequences.

2.2 Model comparisons

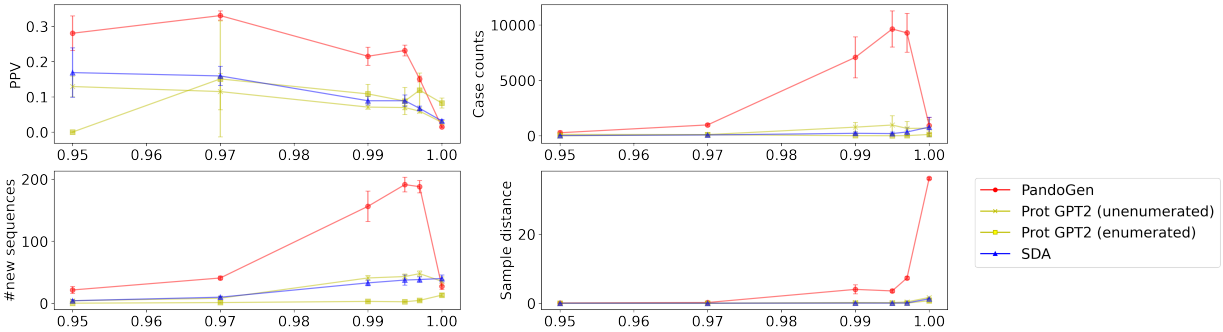

Figure 2: Sequence sample characteristics from the competing methods. PPV or Positive Predictive Value is calculated as the number of novel real sequences divided by the number of novel sequences in the sample, where novel sequences are sequences not reported in the training period. Case counts indicates the number of times novel sequences in the sample are reported in the GISAID database. "#new sequences" refers to the number of novel sequences in the sample that are found in GISAID. Sample distance indicates the k-mer difference between the sample and the training set. The spreads represent the 95% confidence intervals for the runs.

Figure 2 compares the characteristics of samples generated from the models for a range of values of p. We selected values of p between 0.95 and 1. The lower bound was chosen to be 0.95 because, at 0.95 the models output very few novel sequences (e.g., an average of 31 novel sequences out of 2048 generated sequences for the "Prot GPT2 (unenumerated)" case) and values lower than 0.95 are expected to produce even less novelty in the output, hence being uninteresting operating points. We

look at the Positive Predictive Value (PPV) of novel sequences from the models, which is the fraction of real sequences out of generated non-training sequences, the case counts (number times a sequence is reported in GISAID) of non-training sequences, the number of new, real, sequences produced by the models, and the sample difference as detailed above. At $p = 0.95$, we see that most methods produce very few novel sequences as mentioned above. Yet, the PPV at this point is relatively high for most cases. This is because, the models largely regenerate training sequences at this point. As p increases, the number of new sequences which are also real sequences produced by the models starts to increase, and the number produced by PandoGen shows the highest increase. PandoGen produces almost $4\times$ the number of new sequences compared to the second-best case in the best case. PandoGen asserts a bigger lead, close to $10\times$, for case counts. In general, PandoGen produces samples with higher PPV than other methods. PandoGen's performance at $p = 1$ shows a stark drop-off. This is because, at $p = 1$, PandoGen shows a stark increase in the sample difference indicating that at this operating point, the model is trying to predict sequences very different from training sequences, which may only be expected far into the future, which can be difficult and unreliable (hence sample k-mer difference can be used as a method to filter out samples which are too ambitious and error-prone). For $p \geq 0.99$, PandoGen samples show a higher sample distance than competing methods. This may be because PandoGen has explicit constructs in training that prevents the models from regenerating the training set, forcing generated samples to have more differences, compared to standard likelihood-maximization which does not have such a mechanism in training.

Below, we examine some sequence generation metrics as a function of sequence rank. As generative sequence models, all methods assign probabilities to generated samples. Sequence ranking is determined based on this probability value. Higher ranking sequences are expected to have better quality, and focusing on higher ranking sequences is a way to get the most reliable data out of a sample set from the point of view of downstream analyses.

Figure 3 shows the fraction of real sequences among the top-1000 novel sequences generated from the models (or the PPV of novel sequences), as a function of sequence rank. A sequence is real, if it was reported in the GISAID database at any point in time. If a sequence is real but first reported in the training period, it is not considered for the plot. Higher values of PPV indicate lower levels of hallucination [JLF+23], and higher trustworthiness of the model. As may be seen, multiple PandoGen configurations far outperform the other methods. Sampling configurations $p = 0.995$ and $p = 0.997$ lead other configurations.

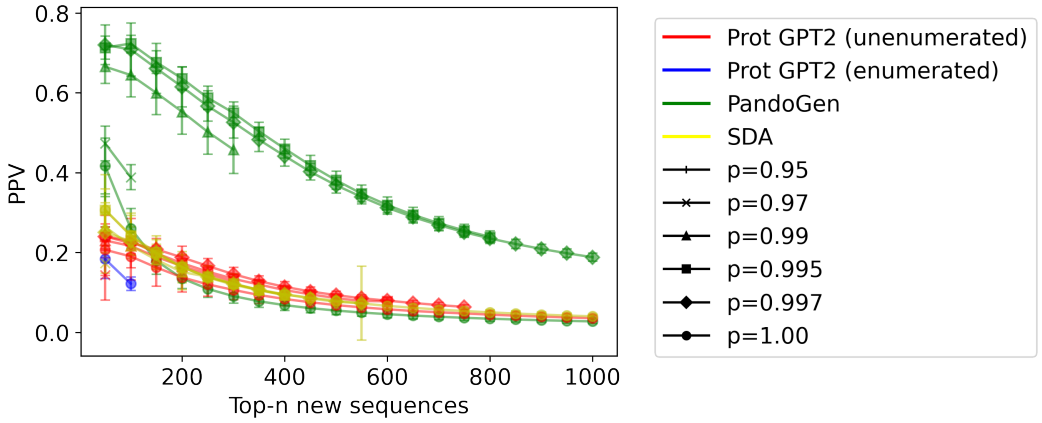

Figure 3: PPV by sequence rank for the top 1000 new sequences produced by the different methods. Error bars represent the 95% confidence intervals for six independent sampling runs.

Figure 4 shows the cumulative number of GISAID cases of the generated novel sequences from the models (sequences not reported in the training period), as a function of sequence rank. Multiple PandoGen configurations dominate the plot, with $p = 0.995$, and $p = 0.997$ coming out winners again, with close to 10 times the case count of competing methods for top-1000 sequences. The curves flatten out further out from the origin, indicating that the more likely sequences from PandoGen are also more potent. This trend holds similarly for other methods as well, but the curves flatten pretty early, whereas PandoGen case counts continue to increase for lower-ranked sequences as well. This also shows

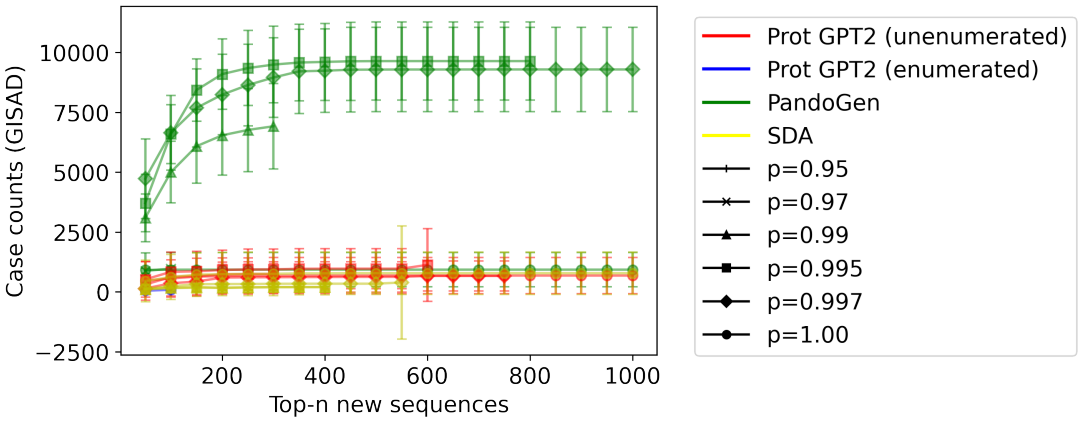

Figure 4: GISAID case count by sequence rank for the top 1000 new sequences produced by the different methods. Error bars represent the 95% confidence intervals for six independent sampling runs.

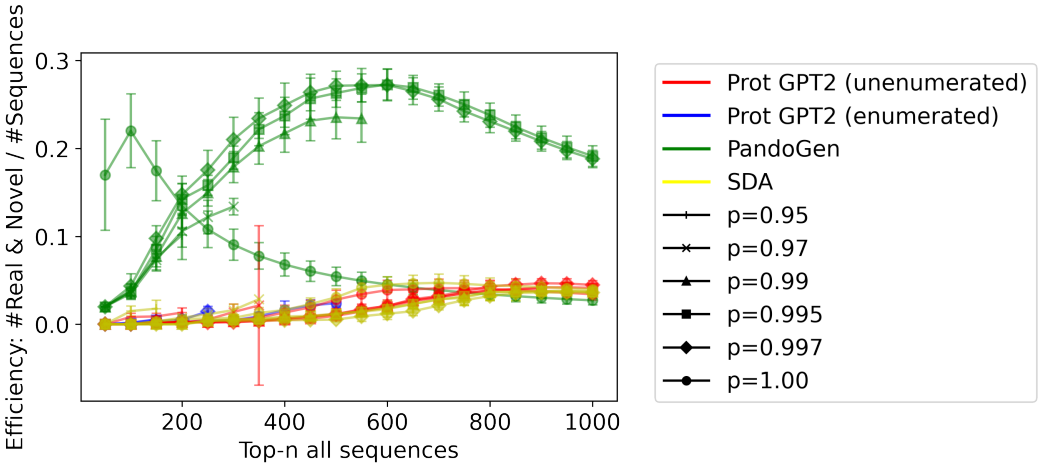

Figure 5: Fraction of real, novel sequences among all sequences generated from the models by sequence rank. Error bars represent the 95% confidence intervals for six independent sampling runs.

that the salience of sequences from PandoGen is higher, as the case count increase is not proportional to the increase in the number of novel sequences from PandoGen, but far higher.

In the last two figures, we looked at stats for novel sequence generations from the model. We next look at the fraction of novel, real sequences produced from the models among *all* sequences generated from the models in Figure 5. This plot indicates that since other methods are trained on the training set, they are not able to produce novel, real sequences at as high a rate as PandoGen. During PandoGen training, we explicitly forced the model to learn to generate sequences not in the training set, and added guardrails through both the Reward Model, as well as a KL-divergence loss term to prevent it from deviating so far from the training set that the outcome is outright wrong.

We note based on Figure 2 that the sample differences of PandoGen's outputs are higher than that of competing methods. The question naturally arises as to whether the improvement in PandoGen's performance is purely due to its access to operating points that are not available to the other models. It is possible to expand the operating range of deep autoregressive models through temperature-shaping of the output distribution [AHS85], which has been employed widely in literature [FG17] [CCF+18] [LCC+22]. Recall that the last layer of a deep autoregressive model produces the distribution of the next sequence element to be generated (Equation 1). In the normal case, the distribution is written as follows.

$$P(x_l = i \mid X_{1:l-1}) = \frac{\exp(f^i(X_{1:l-1}))}{\sum_j \exp(f^j(X_{1:l-1}))} \tag{6}$$

Here f^i are Neural Network outputs. Application of temperature changes the output distribution at each time-step of the sampling process as follows.

$$P(x_l = i \mid X_{1:l-1}) = \frac{exp(\frac{f^i(X_{1:l-1})}{T})}{\sum_j exp(\frac{f^j(X_{1:l-1})}{T})} \tag{7}$$

Values of $T > 1$ flatten the output distribution and increase its entropy beyond the normal operating range.

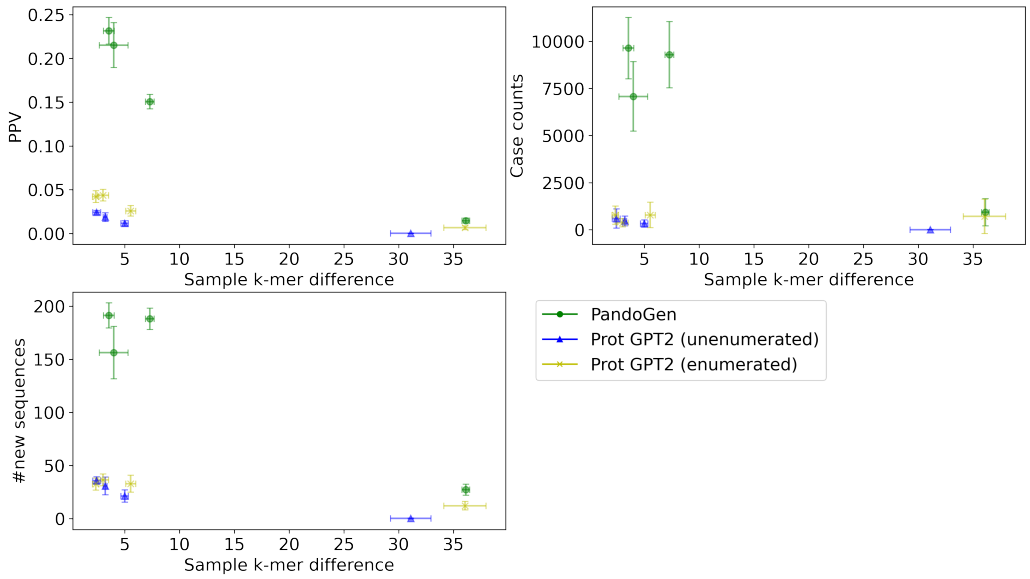

Figure 6: Sequence sample characteristics comparing Prot GPT2 with temperature shaping and PandoGen. Error bars represent the 95% confidence intervals for six independent sampling runs.

Applying this technique allows us to widen the operating range of a competing method, Prot GPT2 for instance. To explore whether a wider operating range allows Prot GPT2 to match PandoGen, we used binary search to locate values of T which allowed samples generated by Prot GPT2 to have similar sample differences as PandoGen's (for $p \geq 0.99$; for these values of p, PandoGen shows higher sample difference compared to others; Figure 2). For each resulting operating temperature, we ran six independent sampling runs to obtain performance characteristics. These characteristics are plotted in Figure 6. As may be seen, PandoGen continues to substantially outperform Prot GPT2 even when enhanced with temperature scaling, indicating that the gains made by PandoGen are not due to access to a wider operating range.

3 Related Works

There exist many analyses and characterizations of SARS-CoV2 sequences using mathematical models and Machine Learning. These methods offer valuable insights with predictive potential. For instance, in a previous study [HZBB21], the authors presented a method to predict point mutations in the Spike protein with potential for viral escape using machine learning techniques from Natural Language Processing (NLP). Mutations in the Spike protein have been characterized on their effect on binding between the ACE-2 receptor and the Spike protein [LZ22], drawing useful insights connecting ACE-2 binding propensity to the potency of the virus. Deep Mutational Learning [TWG+22], uses machine learning and deep sequencing to produce models that can predict the effect of one or more mutations in the receptor binding module (RBM) of the Spike protein to its binding affinity to both the human

ACE-2 receptor as well as to antibodies, offering a detailed picture of the potency of the mutations. In another work [MBW+22], the authors used multiple features including epidemiological, evolutionary, and neural-network-based features to predict the potential of existing individual mutations to become drivers of future Variants of Concern (VoCs). Pyro [OJB+22] is a tool that applies multivariate logistic regression on circulating SARS-CoV2 lineages and their prevalence characteristics, to continually forecast the growth of recent lineages, and predict important mutations with public health implications in existing sequencing data. These and other studies available today lay emphasis on evaluation of the effects of mutations, rather than generation of novel, complete, instances of the Spike protein sequence itself, which is the focus of this article.

Two classes of PLMs exist in literature today. One class, called Masked Protein Language Models (MPLMs) are trained to predict a set of masked locations in a sequence. This allows the models to learn structural properties of the sequence [RMS+19], and can be finetuned for specific applications using application-specific datasets. Popular ways to use masked models for inferencing are to use the embeddings from the model as features for downstream applications, or to directly finetune the model for classification purposes. During inferencing with the model, it is possible to mask certain locations in a sequence and ask the model to predict the missing locations. In this way, theoretically, all possible point mutations and combinations thereof can be explored using the model. However, it is infeasible in practice to use it as a generative model because (1) only point mutations can be modeled whereas the space of sequences contain insertions and deletions as well and (2) the number of possible masks of a sequence scales exponentially as the length of the sequence, resulting in prohibitive computational costs. Notwithstanding these limitations, ProtFound [CNW+22] tries to model a specific segment within the Spike protein using an MPLM involving the generation of millions of masked sequences. Due to the computationally heavy nature, the pipeline uses a supercomputing cluster to run. The work also doesn't treat the efficacy aspect - as in how many of the sequences produced this way are realistic or found in nature. To scale to the length of the full Spike sequence instead of the segment treated in ProtFound [CNW+22] would need tremendous resources and is likely infeasible as the full Spike sequence is much larger than the segment modeled in that work. Our method effortlessly solves the problem of full Spike sequence generation on a single compute node within a limited sequence budget.

The second class of PLMs is the type that is used in this work, the deep autoregressive models that were discussed in Section 1. These are generative models of sequences, and can be used as sampling beds for *de novo* sequence generation, without the restrictions faced by MPLMs. These models are usually initially pre-trained on large corpora of unlabeled datasets through simple likelihood maximization methods, and further customized for a specific task through transfer learning [RSR+20] by finetuning the model on a task-specific dataset also through likelihood maximization. As detailed before there exist multiple examples of deep autoregressive models used to model biological sequences [MKG+23] [FSH22] [SRK+21]. These methods are all trained, and in two cases, finetuned [MKG+23] [Dho23], using standard likelihood maximization. As outlined in Section 1, this approach has many limitations when it comes to generating sequences in a pandemic situation, and our goal is to introduce training methods to counter these issues to som extend.

Related to our work are methods in Natural Language Processing such as Reinforcement Learning from Human Feedback [CLB+23]. Recently, a method called Quark [LWJ+22] was proposed as a drop-in replacement for Reinforcement Learning for the same class of problems. Quark was originally intended to reduce toxicity in generated text using human labels for LLMs that respond to human input. We modify and adapt the solution structure of Quark for performing model updates within the PandoGen training loop as shown in Figure 1.

4 Methods

4.1 Model architecture

We base our models on decoder-only autoregressive models, which consist of stacks of modules, each module consisting of a self-attention layer, a layer-normalization layer, a non-linearity and dropout layer [VSP+17]. The model has eight of these modules, with attention head dimensionality of 128, with 12 attention heads, and a fully connected layer that is three times the attention size. On top of the 8 stacked modules, we have a final Softmax layer that predicts a distribution over the protein

sequence elements (amino acids). We call this final layer, the Protein Model head (PM head), for ease of reference later. Overall, there are approximately 192 million parameters in our models.

Given a training set, each amino acid character in a protein sequence in the training set is converted to an embedding; hence the input protein sequence becomes a sequence of embeddings of the same length. To this sequence, we prepend a special character indicating start of sequence. To this sequence of embeddings, a second embedding sequence of the same length and dimensionality is added. This second embedding indicates the position of each embedding in the sequence, and is trained along with the rest of the model.

The model predicts the probability of a protein sequence as follows.

$$\log P[X_{1:N}] = \sum_i \log P(x_i \mid X_{1:i-1}; \Phi) \tag{8}$$

Here, Φ represents the parameters of the model, and $P(x_i \mid X_{1:i-1}; \Phi)$ is the output of the PM head.

During data generation from the model, the generation is conditioned on the start of sequence character, that is, we ask the model to complete the sequence provided only the start of sequence special character. This deviates from applications of LLMs where the models are usually prompted by some user input instead of simply the start of sequence character, as the LLMs are responding to some cue.

4.2 Pretraining on UniProt dataset

We downloaded the UniProt version $2019 - 09$ which contains protein sequences released before the onset of the COVID-19 pandemic. We first pre-traiend our 192M parameter model using UniRef50 sequences, split randomly into 35M training sequences and 3.9M validation sequences. Pretraining was done for our models using snippets of UniRef50 sequences of length 255. Where a sequence was shorter than length 255, the complete sequence was used and when sequence was longer, a random substring of length 255 was sliced out. During validation, to keep the validation set deterministic, we used only sequences shorter than length 256. We trained the model for two epochs with a learning rate of 1×10^{-4}, and a weight decay of 1×10^{-2}. We employed learning rate warmup for 1024 batches, and thereafter, a linear decay in learning rate was instituted. We used 4 NVIDIA V100 GPUs to pretrain the model using half-precision floating point with a batch-size of 30 per device. Validation loss was calculated once every quarter of an epoch, and the model version with the best validation loss was retained for further finetuning. Pretraining was done for two epochs.

4.3 Finetuning on SARS-CoV2 Spike protein sequences

We downloaded SARS-CoV2 mutation lists (sequences expressed as a list of mutations) from GISAID. To create the training dataset, we selected a cutoff date, and sequences on or before this cutoff date were used for training. The goal would be to predict sequences after this cutoff date.

In the training set, we discarded sequences indicating a stop codon. The remaining sequences were split into training and validation sequences based on a random split of the pango lineages [RHO+20] ssigned to the sequences. Any Spike sequence occurring in both a training and validation pango lineages was moved to the training set.

We finetuned the model resulting from Section 4.2 as well as the Prot GPT2 model, which has also been trained on UniRef50, on this same training/validation split, both for 24 epochs, checkpointing the models four times in each epoch. Both models used a fine-tuning learning rate of 1×10^{-5}, with a weight decay of 1×10^{-2}, 1024 learning rate linear warmup steps and linear decay afterwards. We evaluated each checkpoint on the validation set, and selected the one with the best loss.

We term the finetuned version of our UDA model as the SARS-CoV2 deep autoregressive model (or SDA model) in the sequel, and the Prot GPT2 finetuned model is referred to as "Prot GPT2 (unenumerated)" as the model is finetuned on a unique set of SARS-CoV2 sequences without information about prevalences of different sequences. Both are from a similar family of models, and expected to have similar performance characteristics. However Prot GPT2 is a much larger model with close to 800 million parameters.

In addition, we trained "Prot GPT2 (enumerated)" where the training sequences are not uniquified. So, if a sequence is deposited in GISAID 500 times, the sequence occurs in the training set 500 times

as well. The training data size for this case resulted in a similar number of gradient updates for a single epoch as 24 epochs of the uniquified data used for finetuning the other models. Hence training in this case was performed for a single epoch only.

After training iterations are completed, all model checkpoints are validated against the validation set, and the best-performing checkpoint is chosen for each model.

4.4 Reward model training

The reward model's goal is to predict potential infectiousness for an input sequence as a floating point value. We would like to use the number of times a Spike protein sequence has been reported in the GIAB database as a signal indicating the infectiousness of the sequence. However, it is not straightforward to use this data due to the following reasons. First, a sequence discovered earlier during the pandemic has had more time to spread, compared to a newer sequence. Second, sequencing resources may have needed time to ramp up around the world. Third, restrictions and interventions against the pandemic have shown variation over time. As a result of these, using the raw GISAID case counts as a label is misleading. For instance, the original strain of the SARS-CoV2 virus discovered in 2019 has had many years to circulate, whereas a recently discovered sequence has had very little time, but their case counts may imply that the original strain is more infectious, which is not necessarily true.

To circumvent these challenges, we simulate a game between two sequences. The arena for the game between Seq A and Seq B, is the set of all GISAID cases within the training period that belong to Seq A and Seq B. The game is to randomly select an item from this set. Seq A wins if the selected item is Seq A, and Seq B wins otherwise. The probability of Seq A winning a game is $\frac{N_A}{N_A+N_B}$ where N_A, N_B are the number of Seq A and Seq B GISAID cases in the training period. We ensure that the games are played only between sequences whose first reported date in GIAB is within a narrow timeframe of each other. Implicit here is the assumption that this ensures that they were under similar constraints throughout their lifetimes within the training period. In this *fair* comparison context, relative GIAB case counts may be considered fair estimates of relative potential infectiousness.

Examples are illustrated in Figure 7. Sequences can have a small difference between their respective discovery dates (discovery dates are dates on which the sequences are reported in GISAID), depending on how clear the winner is, between the two sequences. The largest difference in sequence discovery date in our dataset is 5 weeks, but this requires one sequence to have at least 10^5 times more cases than the other. As the ratio of case counts between sequences shrinks, the allowed gap between their discovery dates is also tightened. For example, Sequence A vs Sequence C is an invalid competition because they have very similar case counts, but C was reported two weeks later. When two sequences with similar case counts were first reported in the same week, they are not always compared (Sequence D vs Sequence E). To accept these competitions, we require that the relative case counts of the two sequences are similar in two disjoint geographic regions (Eurasia vs rest of the world in our case), with no change in the predicted winner in majority of the games. Finally, we do not use any sequences first reported in the last 7 weeks of the training period (though we use the case counts of sequences reported earlier in that period), as the sequences reported in this time frame may not have had sufficient time to spread.

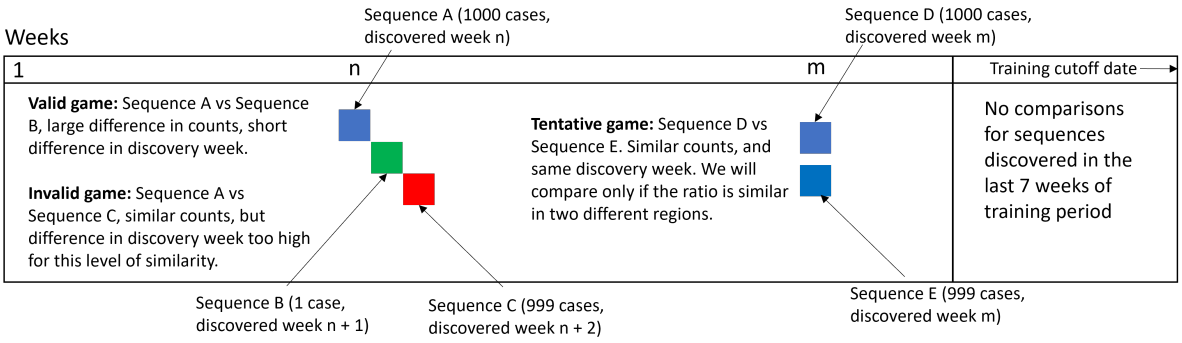

Figure 7: Heuristics for creating *fair* games. We allow games only among sequences first reported within a certain timeframe of each other.

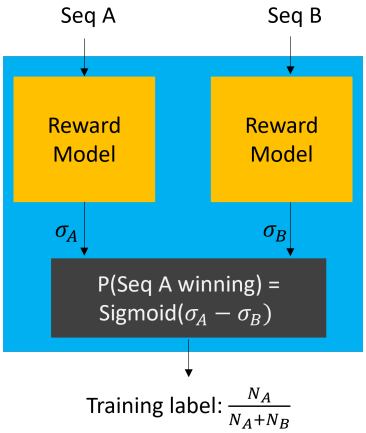

Figure 8: Competition training method. Within the competition model (blue box), the reward model processes two incoming sequences Seq A and Seq B, and produces infectiousness potentials σ_A, σ_B for these respectively. The competition model converts these to probabilities for competition outcomes in a differentiable manner. The probabilities are trained on the ground-truth label. The probability of Seq A winning is trained on the label value $\frac{N_A}{N_A+N_B}$ where N_A is the GISAID case count for sequence A within the training period and N_B is the GISAID case count for the sequence B within the training period.

Table 1: Ratio of case counts of the lower-occurrence sequence to that of the higher-occurrence sequence and allowed tolerance in their weekly discovery dates for them to be allowed to be compared

Case count ratio (r)	Discovery week tolerance
$0 \leq r < 1e-5$	5
$1e-5 \leq r < 1e-4$	4
$1e-4 \leq r < 1e-3$	3
$1e-3 \leq r < 1e-2$	2
$1e-2 \leq r < 1e-1$	1
$1e-1 \leq r < 1$	0

The conditions for two sequences to be allowed to be compared is shown in Table 1. As shown, as the case count ratio between two sequences increases, they are allowed to be discovered further apart in time to be inducted into the competition draw. We apply additional heuristics so that the number of comparisons are tractable. Each sequence can be compared to a maximum of 750 sequences in competitions (random subsampling is used to cut down larger lists). A maximum of 2,000,000 sequences are randomly sampeld from the resultant draw. After this, we run some filtrations. First, sequence pairings whose majority winner changes in two disjoint geographical regions (Eurasia vs non-Eurasia) are removed. Second, sequence pairs where the majority winner's win proportion differs in the two separate geographic region by more than 0.25 are discarded as well. Finally, if the total number of surviving competitions exceeds 1,000,000 competitions, we subsample the list to 1,000,000 cases.

We created a competition model to predict the outcome of these games (Figure 8). The model takes two sequences, Seq A, and Seq B, and predicts the probability distribution of the outcome. Within the competition model is the reward model which predicts an infectiousness potential for Seq A and Seq B, (σ_A, σ_B respectively) and the ensconcing competition model uses the predicted infectiousness potential to predict the outcome distribution. The full model is trained on the *ground truth* outcome distribution calculated from GISAID case counts within the training period as described above. The training objective is to reduce the KL-divergence between the predicted and the label distributions.

The reward model uses the same architecture as the SDA model, except that the PM head is replaced with a reward head. A PM head produces as many outputs as there are characters in our amino acid alphabet, whereas the reward head produces a single scalar output. Also, the reward head

output is only valid after the complete sequence has been passed through the model. The rest of the layers in the reward model are initialized from the SDA model's parameter values.

The competition model is trained with a learning rate of 1×10^{-5} over 1 epoch. A total of four checkpoints were kept for the training. For selecting the checkpoint for downstream use, we used sequences discovered in the 7th week from the end of the training period (note that these sequences were not used in training). Sequences having an occurrence count of over 50 in the last 7 weeks of the training period were labeled 1 and the remaining sequences were labeled 0. The sequences were each scored by the checkpoints, and the area under the receiver operating characteristic (AUROC) curve for binary classification using the scores was computed. The AUROC is a popular measure of classifier efficacy. The checkpoint performing the best was chosen.

4.5 PandoGen finetuning

The final step in our pipeline is to finetune the SDA model directly within its operational setting of sequence generation, using the reward model we presented in Section 4.4. To finetune the model, in addition to the reward signal, we also use a second signal indicating whether a sequence generated by the model already exists in the list of known sequences or not.

To perform this finetuning, we adapt an algorithm called Quark [LWJ+22]. Quark was developed to improve an LLM's ability to generate text that conforms to certain qualities such as reduced toxicity. We prefer Quark to Reinforcement Learning [CLB+23] [SOW+20] because it has been reported to be stabler, and easier to implement using libraries created for LLM training.

To explain PandoGen training, first we recollect (Section 4.1) that the SDA model generates sequences conditioned on a special beginning of sequence character. The PandoGen model, which is a finetuned version of the SDA model, generates sequences conditioned also on a second special character. This second special character corresponds to sequence quality. We call this, the reward token. During training, we generate sequences from PandoGen conditioned on the *highest quality reward token* (to be explained in a moment). The generated sequences are scored using the reward model, and then looked up in the set of historically known sequences. If the sequence is historically known, we assign it a special reward token indicating historicity. If not, we assign it the reward token associated with the quantile within which the reward for the generated sequence lies. The quantiles are determined based on generations from the SDA model scored by the reward model. Hence, the highest quality reward token is assigned to sequences with rewards in the highest quantile, but which are not sequences in the original training set. The generated sequences annotated with the reward tokens are added to a data pool, $\mathcal{D}$ of sequences (initialized using data generated from the SDA model). A fixed-size sample, D, is taken from this data pool, and used to perform a finetuning step. The finetuning step maximizes the following objective through mini-batch gradient descent.

$$E_{X \sim D} \sum_i \Big(\log P(x_i \mid X_{1:i-1}, r_X; \Phi) - \beta D_{KL}(p(x_i \mid X_{1:i-1}; \Phi_{SDA}) \| p(x_i \mid X_{1:i-1}, r_X; \Phi)) \Big)$$

Here, Φ_{SDA} represents the parameter set of the original SDA model which is not adjusted during PandoGen finetuning, and Φ represents the parameter set of the PandoGen model, which is initialized from Φ_{SDA}, and modified through the finetuning step. r_X is the reward token for the sequence X, assigned based on the procedure mentioned above. Note that this quantity is slightly different from the original Quark cost function, which requires D to be a stratified subset of $\mathcal{D}$, where sequences in each reward quantile are uniformly represented. We found the non-stratified case to achieve higher rewards. In addition, we also noted that not having dropout also helps faster convergence to higher reward solutions during training. Hence, we set dropout to 0 for our PandoGen finetuning stage.

The procedure of generating sequences from the PandoGen model, scoring it using the reward model, checking historicity, and finetuning the model using the annotated generations is carried out iteratively. Through the procedure, the PandoGen model is trained to generate sequences according to their quality and historicity (via conditioning on the reward token, r_X). It learns to generate high reward, non-historic sequences when conditioned on the highest quality reward token. At the same time, the loss term $\beta D_{KL}(p(x_i \mid X_{1:i-1}; \Phi_{SDA}) \| p(x_i \mid X_{1:i-1}, r_X; \Phi))$ prevents the model from deviating too far from the probability distribution learnt by the original SDA model. This is important as the SDA model has a grasp of the structure of the SARS-CoV2 Spike protein sequence, and deviating too far from the basic structure is detrimental to our goals. The training scheme is shown in Figure

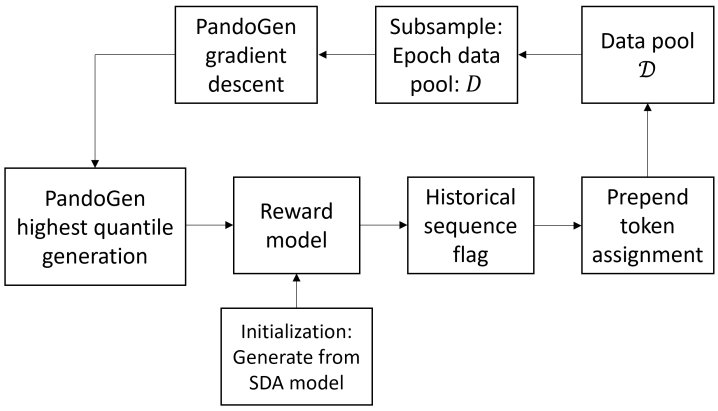

Figure 9: The PandoGen finetuning procedure follows three steps: sequence generation, sequence scoring/categorization, and finetuning using the scored/categorized sequences

9. To determine when to stop training we use the early stopping criterion, whereby if the reward of the generated batches from the model doesn't increase for three consecutive training epochs, training is halted and the epoch with the highest reward is chosen.

5 Conclusion

We presented a novel method to train deep generative models to generate future pandemic sequences with the goal that the generated sequences must have a high fraction of real sequences, and that the case count of the generated corpus should be high. We compared our training pipeline with a standard model in the field and found that our method, outperforms the existing method by 4 times in terms of number of novel sequences and close to 10 times for case counts of the generated corpus. Our training pipeline used only information available in GISAID, and did not depend on additional laboratory experiments to perform sequence characterization.

It is of vital importance that we prepare for the next pandemic. Information-sharing systems such as GISAID were setup very quickly, to combat the COVID-19 pandemic. We envisage a similar system of information exchange in the unfortunate event that another pandemic arises in the future. These systems would only respond better in a future scenario, with sequences deposited sooner with a more well-prepared sequencing infrastructure around the world with faster and cheaper sequencing technology. Our work is aimed at taking advantage of such information sharing. We hope methods such as ours, will provide valuable tooling supporting pandemic management.

6 Acknowledgement

This work utilizes resources supported by the National Science Foundation's Major Research Instrumentation program, grant #1725729 [KMZ+20], as well as the University of Illinois at Urbana-Champaign.

References

[AHS85] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

[CCF+18] Massimo Caccia, Lucas Caccia, William Fedus, Hugo Larochelle, Joelle Pineau, and Laurent Charlin. Language gans falling short. *arXiv preprint arXiv:1811.02549*, 2018.

[CLB+23] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences, 2023.

[CNW+22] Jie Chen, Zhiwei Nie, Yu Wang, Kai Wang, Fan Xu, Zhiheng Hu, Bing Zheng, Zhennan Wang, Guoli Song, Jingyi Zhang, et al. Running ahead of evolution-ai based simulation for predicting future high-risk sars-cov-2 variants. *bioRxiv*, pages 2022–11, 2022.

[DCR+23] Bernadeta Dadonaite, Katharine HD Crawford, Caelan E Radford, Ariana G Farrell, C Yu Timothy, William W Hannon, Panpan Zhou, Raiees Andrabi, Dennis R Burton, Lihong Liu, et al. A pseudovirus system enables deep mutational scanning of the full sars-cov-2 spike. *Cell*, 186(6):1263–1278, 2023.

[DEKM98] Richard Durbin, Sean R Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

[Dho23] Rahul M. Dhodapkar. A deep generative model of the sars-cov-2 spike protein predicts future variants. *bioRxiv*, 2023.

[DSM+21] Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. *arXiv preprint arXiv:2104.08758*, 2021.

[EHD+21] Ahmed Elnaggar, Michael Heinzinger, Christian Dallago, Ghalia Rehawi, Yu Wang, Llion Jones, Tom Gibbs, Tamas Feher, Christoph Angerer, Martin Steinegger, 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.

[FF14] Douglas M Fowler and Stanley Fields. Deep mutational scanning: a new style of protein science. *Nature methods*, 11(8):801–807, 2014.

[FG17] Jessica Ficler and Yoav Goldberg. Controlling linguistic style aspects in neural language generation. *arXiv preprint arXiv:1707.02633*, 2017.

[FSH22] Noelia Ferruz, Steffen Schmidt, and Birte Höcker. Protgpt2 is a deep unsupervised language model for protein design. *Nature communications*, 13(1):4348, 2022.

[GPAM+20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[HBD+19] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

[HMRC21] Yun Heo, Gowthami Manikandan, Anand Ramachandran, and Deming Chen. Comprehensive evaluation of error-correction methodologies for genome sequencing data. *Exon Publications*, pages 89–108, 2021.

[HZBB21] Brian Hie, Ellen D. Zhong, Bonnie Berger, and Bryan Bryson. Learning the language of viral evolution and escape. *Science*, 371(6526):284–288, 2021.

[JLF+23] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, 2023.

[KMZ+20] Volodymyr Kindratenko, Dawei Mu, Yan Zhan, John Maloney, Sayed Hadi Hashemi, Benjamin Rabe, Ke Xu, Roy Campbell, Jian Peng, and William Gropp. Hal: Computer system for scalable deep learning. In *Practice and experience in advanced research computing*, pages 41–48. 2020.

[LCC+22] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

[LWJ+22] Ximing Lu, Sean Welleck, Liwei Jiang, Jack Hessel, Lianhui Qin, Peter West, Prithviraj Ammanabrolu, and Yejin Choi. Quark: Controllable text generation with reinforced unlearning. *arXiv preprint arXiv:2205.13636*, 2022.

[LZ22] Zhendong Li and John ZH Zhang. Mutational effect of some major covid-19 variants on binding of the s protein to ace2. *Biomolecules*, 12(4):572, 2022.

[MBW+22] M. Cyrus Maher, Istvan Bartha, Steven Weaver, Julia di Iulio, Elena Ferri, Leah Soriaga, Florian A. Lempp, Brian L. Hie, Bryan Bryson, Bonnie Berger, David L. Robertson, Gyorgy Snell, Davide Corti, Herbert W. Virgin, Sergei L. Kosakovsky Pond, and Amalio Telenti. Predicting the mutational drivers of future sars-cov-2 variants of concern. *Science Translational Medicine*, 14(633):eabk3445, 2022.

[MKG+23] Ali Madani, Ben Krause, Eric R Greene, Subu Subramanian, Benjamin P Mohr, James M Holton, Jose Luis Olmos Jr, Caiming Xiong, Zachary Z Sun, Richard Socher, et al. Large language models generate functional protein sequences across diverse families. *Nature Biotechnology*, pages 1–8, 2023.

[ODZ+16] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[OJB+22] Fritz Obermeyer, Martin Jankowiak, Nikolaos Barkas, Stephen F Schaffner, Jesse D Pyle, Leonid Yurkovetskiy, Matteo Bosso, Daniel J Park, Mehrtash Babadi, Bronwyn L MacInnis, et al. Analysis of 6.4 million sars-cov-2 genomes identifies mutations associated with fitness. *Science*, 376(6599):1327–1332, 2022.

[Ope23] OpenAI. Gpt-4 technical report, 2023.

[OWJ+22] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.

[RHO+20] Andrew Rambaut, Edward C Holmes, Áine O'Toole, Verity Hill, John T McCrone, Christopher Ruis, Louis du Plessis, and Oliver G Pybus. A dynamic nomenclature proposal for sars-cov-2 lineages to assist genomic epidemiology. *Nature microbiology*, 5(11):1403–1407, 2020.

[RMS+19] Alexander Rives, Joshua Meier, Tom Sercu, Siddharth Goyal, Zeming Lin, Jason Liu, Demi Guo, Myle Ott, C. Lawrence Zitnick, Jerry Ma, and Rob Fergus. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *PNAS*, 2019.

[RNS+18] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[RPG+21] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation, 2021.

[RSR+20] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[RWC+19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[SM17] Yuelong Shu and John McCauley. Gisaid: Global initiative on sharing all influenza data–from vision to reality. *Eurosurveillance*, 22(13):30494, 2017.

[SOW+20] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021, 2020.

[SRK+21] Jung-Eun Shin, Adam J Riesselman, Aaron W Kollasch, Conor McMahon, Elana Simon, Chris Sander, Aashish Manglik, Andrew C Kruse, and Debora S Marks. Protein design and variant prediction using autoregressive generative models. *Nature communications*, 12(1):2403, 2021.

[TWG+22] Joseph M Taft, Cédric R Weber, Beichen Gao, Roy A Ehling, Jiami Han, Lester Frei, Sean W Metcalfe, Max D Overath, Alexander Yermanos, William Kelton, et al. Deep mutational learning predicts ace2 binding and antibody escape to combinatorial mutations in the sars-cov-2 receptor-binding domain. *Cell*, 185(21):4008–4022, 2022.

[uni17] Uniprot: the universal protein knowledgebase. *Nucleic acids research*, 45(D1):D158–D169, 2017.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[Xia21] Xuhua Xia. Domains and functions of spike protein in sars-cov-2 in the context of vaccine design. *Viruses*, 2021.