https://doi.org/10.1038/s41587-023-02115-w

Generative models for protein structures and sequences

Chloe Hsu, Clara Fannjiang & Jennifer Listgarten

Models like ChatGPT and DALL-E2 generate text and images in response to a text prompt. Despite different data and goals, how can generative models be useful for protein engineering?

Natural evolution creates new proteins through stochastic processes such as mutation and recombination, coupled with the screening filter of organismal fitness. In the past few decades, proteins have been engineered with directed evolution, which roughly mimics natural evolution, only doing so with a user-specified filter to select desired properties. Directed evolution typically starts from a known protein and explores only similar sequences, typically by using stochastic laboratory methods for sequence modification. Diversification, naturally occurring or otherwise, can be thought of as a generative mechanism for protein sequence candidates. With recent advances in machine learning, along with advances in modern-day sequence synthesis, it is becoming feasible to use more complex generative mechanisms to explore much broader swaths of protein spaces more efficiently. A wide variety of machine learning modeling strategies can be useful for different problems involving proteins. Herein we focus on generative models—those that can generate protein sequences and/or structures—and particularly on conditional generative models that generate sequences and/or structures that are consistent with a specified property such as a protein family[1], an active-site structure (called functional site scaffolding)[2–6] or a specified backbone structure (called inverse folding)[7–10].

Although conditional generative models for protein sequences and structures are relatively new, they are closely related to unconditional generative models, such as profile hidden Markov models (HMMs) and Potts models. An unconditional generative model learns a probability distribution, $p_\theta(X)$, where the values in θ are the learned parameters of the generative model. Implicitly, the model represents a distribution for just one condition. For example, a Potts model for one protein family learns the distribution of amino acid sequences, X, of proteins in that family. When there are sufficiently many evolutionarily related homologs for the protein of interest, such unconditional generative models can be a simple, fruitful strategy for generating diverse protein sequences.

In contrast to their unconditional counterparts, conditional generative models learn not only one probability distribution for one condition, but rather, a collection of many probability distributions, one for each of many conditions. In principle, the condition in a conditional generative model can be any property, specified as a one-dimensional scalar, a vector, a matrix or so forth. These may or may not contain geometric information such as 3D structure. The generative output for the protein could be encoded as a sequence, matrix or graph. For simplicity, we anchor our discussion on models that generate amino acid sequences or structures in the form of 3D coordinates, but much of our discussion applies equally to other types of generative modeling problems (Fig. 1). More precisely, a conditional generative model maps each condition y to a conditional probability distribution, $p_\theta(X \mid Y = y)$. For each setting, $Y = y$, the model represents a different conditional probability distribution over X. For example, while a backbone-conditional generative model conditions on one specific set of backbone coordinates for each use case, the overall model is trained with the goal that any set of 3D coordinates could be used.

Conditional models need not be retrained for every new condition and might be able to interpolate between conditions they were trained on. By jointly modeling all the conditions, conditional generative models can learn each conditional distribution in a way that borrows relevant information from similar conditions. For example, when applying a conditional generative model for fixed backbone sequence design, we may not have trained on examples of the exact backbone structure of interest, but the model may nevertheless have learned from other similar or partially similar backbones, allowing the model to generalize to never-observed backbones. Therefore, use of conditional generative models is particularly compelling when the exact condition of interest was not present, or only rarely present, in the training data, but related conditions are abundant. Critically, the desired generality of conditional generative models means that they correspondingly need to learn from a wide variety of data that sufficiently encapsulate the intended use cases. The choice of training data is a subjective decision that directly influences the resulting model; it expresses our beliefs about which protein distributions we seek to learn. Although practically one can always generate outputs for any specifiable condition, there is no guarantee of correctness for any generated outputs.

Training and sampling from conditional generative models

Machine learning models have free parameters that need to be learned during the training procedure. When we train a model on data, we select ('learn') one specific set of parameter values that is 'best' suited to the training data. To learn the parameters of a model, be it a neural network or a classical statistical model, we must define a loss function that tells us how much we 'lose' by using a particular set of parameters—that is, how 'inappropriate' a parameter setting is for the training data. A classical objective function from statistics is that of the likelihood of the observed data, which, when maximized, yields maximum likelihood estimation (MLE) of the parameters. One appealing aspect of MLE is that it is consistent: as we get increasingly more training data, the estimated parameters yield a distribution that is increasingly closer to the true distribution—assuming that the model class encompasses the true data distribution. In deep learning, MLE is sometimes equivalently described as minimizing the cross-entropy between the observed distribution and the predicted distribution—an information-theoretic notion to measure the difference between two distributions. In sequence generation

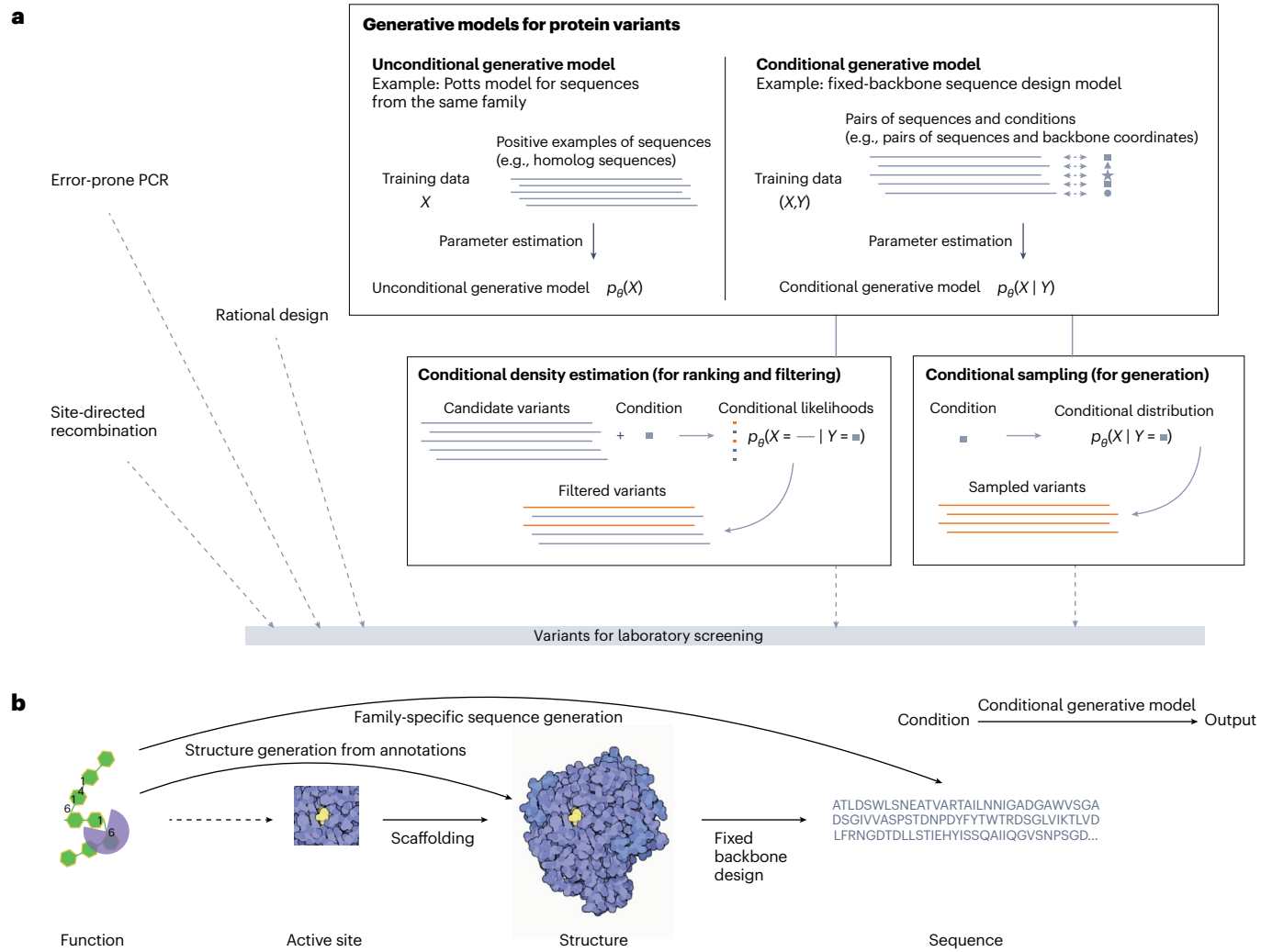

Fig. 1 | Overview of generative models for protein engineering. a, Generation of protein sequences is a core part of protein engineering and other bioengineering tasks. In addition to more classical laboratory generative processes such as error-prone PCR and site-directed recombination, along with physics-based rational sequence design methods, we are now finding that machine learning-based generative models are emerging as a new and powerful sequence generation approach. **b**, Conditional generative models can condition on, and generate, different types of entities in protein engineering, such as generating sequence from structure (such as in fixed-backbone sequence design), sequence from function, or full structure from partial structure (such as scaffolding around a specified active site geometry). Protein structure adapted from D. Goodsell, RCSB PDB, https://doi.org/10.2210/rcsb_pdb/mom_2006_2, 2006.

contexts, MLE is sometimes equivalently described as minimizing the perplexity—a more human-interpretable quantity indicating the average uncertainty at each position. To minimize the loss, we start with some (typically random) initial setting of the parameters and then iteratively refine them using techniques from numerical optimization, typically gradient descent, until the parameters converge according to a predefined stopping criterion, such as when the validation loss changes beneath some threshold. Once model parameters have been learned, we can use the model to conditionally generate protein sequences or structures—that is, to sample from the model to produce proteins that have a high likelihood, assuming the condition has been satisfied. Most learned conditional distributions are non-deterministic (that is, stochastic), meaning that a variety of samples would be generated from one condition. This stochasticity arises from the fact that numerous outputs may satisfy a given condition, and from the uncertainty in the mapping from condition to output. Sequences (or structures) that are more likely (according to the model) to satisfy the condition will have higher conditional likelihoods and are more likely to be sampled. However, by chance alone, we may sample sequences that do not satisfy the condition at all.

Conditional generative models for sequences

Conditional generative models for protein sequences might be set up to condition on the protein function, the protein family[1] or the backbone structure[7–9]. These models are heavily relied upon as a second conditional-generative step after first using a generative model to generate the protein backbone[2,4,5]. One technical challenge in sequence generation, for both conditional and unconditional generative models, is the vast number of theoretically possible sequences: 20^N possible amino acid sequences can be generated for a protein of length N.

Consequently, generative models for sequences are built on strategies to break down both the modeling and the sampling problems into smaller, more manageable parts. A common strategy is to factorize the conditional probability distribution—that is, decompose the likelihood into separate factors, where each factor is itself manageable and all factors can be easily combined by multiplying them together. A classic example of a class of unconditional models are profile hidden Markov models (HMMs), used for sequence alignment, classification and generation. These models assume that the amino acid or nucleotide at one position depends only on the previous position (or insertion/deletion), thereby making it a autoregressive model. HMMs as simple autoregressive models yield algorithms for training and sampling that scale linearly in the length of the sequence. However, many problems require more sophisticated dependencies between amino acids at different positions for accurate modeling. These more complex dependencies can be captured, for example, by composing multiple modeling layers together, such as in Transformer-based autoregressive models. These models can better capture long-range dependencies across a sequence. Despite the added complexity, these models, like HMMs, break down the sequence generation problem sequentially, generating one amino acid at a time. In contrast to HMMs, which are a special case of autoregressive models, more general autoregressive models assume that each amino acid may depend on all the previous amino acids.

One limitation of autoregressive models is that, naively, they can only be used to generate sequences in one specific chosen sequential ordering. To address this limitation, one could explicitly train models with different orderings, so-called 'order-agnostic' models, and learn more flexible autoregressive models that can generate sequences in arbitrary orders. The factorization of a conditional distribution need not, and typically does not, reflect any causal mechanism. However, different factorizations or orderings of variables will generally lead to different models, each of which may be more or less useful. Beyond HMMs and more general autoregressive models, Potts models—also known as Markov random fields—are designed to explicitly capture the interaction between all pairs of positions in a sequence. However, Potts models are limited in that only pairwise interactions can be explicitly modeled, and not higher-order interactions. Variational autoencoders, another class of unconditional generative models that can, in principle, model interactions of arbitrarily high order, have also been applied to modeling protein sequences and structures.

While any of the unconditional sequence-generative models could in principle be adapted to serve as conditional generative models, those with an autoregressive output layer tend to be the choice for sequence generation (see also the Primer by Ruffolo and Madani). For example, the 'next-token' language models in the GPT (Generative Pretrained Transformer) series are also autoregressive conditional generative models, where the condition is the initial prompt. In conditional generation, autoregressive models condition on the input at each step of the generation process, adjusting predictions on the basis of both the condition and the portion of the sequence that has so far been generated. Modern-day autoregressive conditional generative models typically employ a neural network architecture to parameterize the probability distribution. Consequently, the same neural network considerations that emerge in other general machine learning settings similarly emerge here. That is, the user must consider architecture choices such as whether or not to use convolution and attention layers, how many layers to use, and the width of each layer. The neural network architecture could be graph-based, could use geometric features, and could encode certain symmetries of the physical world, such as the symmetry that a protein is the same protein no matter how we rotate it in space (that is, equivariant or invariant to 3D rotations). As these neural network modeling considerations depend on the specific problem settings, there is no universal solution, and the choice is guided by domain knowledge, experience, computational cost and empirical comparisons. Having said that, in practice, an encoder–decoder architecture is used frequently for autoregressive conditional generative models. The encoder network encodes the input condition y into a 'hidden' or latent representation, and the decoder network decodes the hidden representation into a conditional sequence distribution. Besides encoder–decoder architectures, decoder-only architectures have also been used for conditional generation, whereby sequences are conditioned directly on a label for the protein family.

Although named 'generative' models, many conditional generative models can also be used for scoring and ranking sequences. To do so, the conditional likelihoods of a set of candidate sequences are computed, yielding a probabilistic score for each candidate sequence. These conditional likelihoods, $p_\theta(X = x \mid Y = y)$, can be used to rank how likely a sequence x is to satisfy the given condition y, in comparison to other possible sequences. This type of scoring and ranking often emerges in zero-shot protein fitness prediction and variant effect prediction.

Conditional generative models for structures

While protein sequences consist of a finite, discrete vocabulary of 20 amino acids, the 3D coordinates of protein structures are real values containing geometric information; consequently, they introduce a different set of modeling challenges and solutions. In contrast to an autoregressive strategy, a common strategy for structure generation is to break the sampling down into an iterative process over all coordinates at the same time, where each sample of all coordinates depends on the entire previous sample. Diffusion models, which adopt this iterative approach, have recently emerged as a powerful class of models for both unconditional and conditional structure generation. Instead of directly estimating a (conditional) probability distribution over structures, diffusion models estimate how the probability density of such a distribution changes in local neighborhoods of the structure space—that is, how the likelihood increases or decreases as the 3D coordinates of the structure are perturbed. An intuitive analogy would be to describe a path that someone took through the streets by listing the new direction they took after every few steps, rather than to list the set of actual map coordinates they traced out—the path can be constructed from the directions and number of steps taken. Formally, diffusion models learn the gradient of the probability distribution rather than the distribution itself. The probability distribution can implicitly be recovered from the gradient information, but learning a distribution in this manner appears to have practical advantages.

At training time, first we iteratively 'diffuse' a protein structure in the training data by gradually adding random noise to it, until it is unrecognizable—that is, until it consists of either random 3D coordinates or 3D coordinates corresponding to an unstructured polymer. Then we train a neural network to learn how to reverse this diffusion process, step by step, such that we can go from the unrecognizable structure, or any intermediate thereof, to the original, well-formed protein structure. In some cases, reverse diffusion models have been learned from scratch, while in others, advances in protein folding models have been repurposed and fine-tuned to perform the reverse diffusion. After the model has been trained, we can generate structures by starting with a random set of 3D coordinates and iteratively

applying the learned reverse diffusion model, each time incrementally adjusting the 3D coordinates to move toward more likely coordinates, according to the implicit likelihood of the model. For a well-trained model, the reverse diffusion should yield a nicely structured protein, having started with random coordinates. While this process may seem analogous to following a force field in molecular dynamics simulations, the diffusion process in diffusion models does not necessarily carry physical meaning.

There are two main strategies to enable diffusion models to be conditional diffusion models, both having been used for protein structure generation. In 'classifier-free guidance'[2,4,10], the generative process that reverses the diffusion process—described by a neural network—takes in a 'condition tag' at the beginning that enables it to tailor the process accordingly, at both training and test time. Such models must be trained directly with the desired class of use-case conditions, as the conditioning is baked into the neural network. In contrast, in 'classifier-guided' conditioning[5], the conditioning does not happen in the neural network; rather, the neural network is paired with an external classifier that has been separately trained to classify, for example, whether a 3D structure is likely to satisfy the condition of interest; consequently, the diffusion process and the conditioning are decoupled. Classifier-guided conditioning thus allows the same unconditional diffusion model to be easily repurposed for a wide range of conditions without retraining it. Consequently, the classifier-guided approach has the potential to accommodate a wide range of different input condition modalities. Of course, the quality of the conditional generation depends critically on the quality of the classifier. Both the classifier-free and classifier-guided conditioning approaches have been applied to protein structure generation tasks, including conditioning on secondary structures, coarse contact maps, partial structural motifs, symmetry and biochemical properties.

Final considerations

To date, protein generative model have typically generated either sequences or structures, but rarely both. A typical workflow would be to first generate a backbone structure conditioned on some desired property and then, conditioned on this generated structure, generate a sequence—where each of these two generation steps uses a separate model, one before the other. This two-step generation process comes with limitations, including the inability to generate more fine-grained structure that includes the positions of all atoms rather than just the backbone atoms, because the number and identities of the side-chain atoms are tied to the choice of the amino acid sequence. Efforts are now underway to do these steps jointly.

Unlike the case for generated images and texts, the average human cannot look at a generated protein sequence or structure and decide whether it meets their needs. Rather, we require expensive and time-consuming experimental work to validate any generated proteins. Hence, generative biology is in a more difficult position than generative modeling for images and texts. The potential rewards for engineering therapeutics and enzymes are correspondingly large. A potential pitfall of conditional generative models is in the sometimes-mistaken belief that once we build such a model, in sampling from it we will achieve the desired proteins. However, these models are only as good as the data and components that make them up, including data containing critical information about the mapping from condition to output, such as from function to sequence. Limited knowledge about functional mappings in particular is perhaps the most wide-open challenge in protein engineering today, one that cannot be overcome by even the largest or most sophisticated models. Indeed, while machine learning has rapidly transformed the field of structure prediction, there remains substantial work to be able to reliably predict protein function. Consequently, it will generally be more difficult to build accurate conditional generative models for functions not primarily driven by macroscopic structure, such as enzymatic activity. For enzymes, generative models now typically condition on an already known active site; this is not redesigned. Functions that are largely determined by relatively macroscopic structure, however, such as binding, are currently in reach. Our ability to predict structure better than function is due in part to the relative dearth of data for protein function, while protein structures are now relatively numerous. Efforts have been made to generate proteins by conditioning on text-based functional annotations, leveraging advances in natural language processing. However, annotations can be imprecise and unreliable, and may be less suitable for helping a model to figure out how to share data across similar scenarios as compared to more fundamental physical descriptors.

Chloe Hsu ✉, **Clara Fannjiang** & **Jennifer Listgarten** ✉
University of California, Berkeley, Berkeley, CA, USA.
✉e-mail: chloehsu@berkeley.edu; jennl@berkeley.edu

Published online: 15 February 2024

References

1. Madani, A. et al. *Nat. Biotechnol.* **41**, 1099–1106 (2023).
2. Watson, J. L. et al. *Nature* **620**, 1089–1100 (2023).
3. Wang, J. et al. *Science* **377**, 387–394 (2022).
4. Krishna, R. et al. Preprint at *bioRxiv* https://doi.org/10.1101/2023.10.09.561603 (2023).
5. Ingraham, J. et al. *Nature* **623**, 1070–1078 (2023).
6. Eguchi, R. R., Choe, C. A. & Huang, P. S. *PLOS Comput. Biol.* **18**, e1010271 (2022).
7. Ingraham, J., Garg, V., Barzilay, R. & Jaakkola, T. In *Advances in Neural Information Processing Systems* Vol. 32 (2019).
8. Dauparas, J. et al. *Science* **378**, 49–56 (2022).
9. Hsu, C. et al. In *Intl Conf. Machine Learning* 8946–8970 (PMLR, 2022).
10. Anand, N. & Achim, T. Preprint at *arXiv* https://doi.org/10.48550/arXiv.2205.15019 (2022).

Acknowledgements

We thank A. Busia, H. Jiang, M. Lukarska, H. Nisonoff, Y. Song and J. Xiong for discussions and feedback.

Competing interests

J.L. consults for Fable Tx and Inscripta. C.H. is a cofounder of Escalante Bio. The remaining authors declare no competing interests.

Additional information

Peer review information *Nature Biotechnology* thanks the anonymous reviewers for their contribution to the peer review of this work.