

Property guidance for protein sequence generative models with ProteinGuide

Received: 23 May 2025

Accepted: 5 June 2026

Published online: 29 July 2026

 Check for updates

Junhao Xiong^{1,6}, Ishan Gaur^{1,6}, Maria Lukarska^{2,3,4,6}, Hunter Nisonoff^{1,6}, Luke M. Oltrogge^{1,2,3,4}, David F. Savage^{1,2,3,4}✉ & Jennifer Listgarten^{1,5}✉

No principled framework exists for conditioning sequence generative models for protein engineering on auxiliary information, such as experimental data, without additional training of a generative model. Here we present ProteinGuide, a method for such ‘on-the-fly’ conditioning. ProteinGuide is amenable to a broad class of protein generative models including masked language models such as ESM3, any-order autoregressive models such as ProteinMPNN and diffusion and flow-matching models on discrete state-spaces such as MultiFlow. ProteinGuide stems from a unifying statistical framework for these model classes. As proof of principle, pretrained generative models are used to design proteins with user-specified properties, such as higher stability or activity. Proteins are additionally designed to optimize for two desired properties that are in tension with each other. Lastly, we apply ProteinGuide jointly with wet-lab data generation to increase the editing activity of an adenine base editor *in vivo*, resulting in a base editor with higher editing efficiency than was previously achieved using seven rounds of directed evolution.

Generative models are enabling protein engineers to create bespoke proteins^{1–3}, with potential application to therapeutic, agricultural and environmental use cases. In these practical design settings, proteins are engineered to display properties that are rare in the distribution of natural proteins. For example, suppose that one wishes to increase the catalytic activity of an enzyme for some substrate; proteins with such a desired property are not likely to be well represented in most general-purpose, pretrained generative models, nor are they likely to be well represented by conditioning on existing annotation labels in such models. Consequently, experimental measurements of sequence–function relationships must be acquired to glean which proteins are more likely to express this property at the desired level. With these newly generated data, a predictive model can be constructed for the measured property and leveraged to generate new sequences to better achieve the design goal. Current strategies for incorporating such a predictive model into sequence generation remain limited and are often ad hoc.

One such strategy is to use post hoc filtering, that is, to generate many candidate sequences from a general, pretrained model and then to filter these sequences according to a predictive model^{4,5}. However, post hoc filtering is fundamentally constrained by the pretrained model’s distribution, which makes sampling of desired proteins increasingly inefficient as the desired sequences become rarer in the pretrained generative model distribution⁶. Additionally, the sampling inefficiency problem is compounded when filtering on more than one property⁷. Alternatively, fine-tuning—which does not modulate the sampling distribution ‘on the fly’ and does require retraining a generative model—is another commonly used approach^{8,9}. In fine-tuning, a set of proteins thought to most exhibit the desired property, such as those that scored most highly in the laboratory, are used to continue training the pretrained generative model on the same loss function^{10–12}. Note that the term fine-tuning is somewhat amorphous; herein, we use it according to the common usage in protein engineering, as described

¹Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, Berkeley, CA, USA. ²Department of Molecular and Cell Biology, University of California, Berkeley, Berkeley, CA, USA. ³Innovative Genomics Institute, University of California, Berkeley, Berkeley, CA, USA.

⁴Howard Hughes Medical Institute, University of California, Berkeley, Berkeley, CA, USA. ⁵Center for Computational Biology, University of California, Berkeley, Berkeley, CA, USA. ⁶These authors contributed equally: Junhao Xiong, Ishan Gaur, Maria Lukarska, Hunter Nisonoff. ✉e-mail: savage@berkeley.edu; jennl@berkeley.edu

above. Fine-tuning concentrates the generative model distribution on the fine-tuning dataset, thereby coercing it to sample from the relevant part of the existing pretrained model distribution, rather than enabling extrapolation beyond that distribution^{10,11,13}. Furthermore, fine-tuning risks overwriting important general domain information learned by the model^{14,15}, such as features related to the stability of a protein. Such information is learned by the pretrained model from datasets that are typically orders of magnitude larger than the fine-tuning dataset.

Altogether, a methodological framework is needed that allows for sampling sequences from a general, pretrained model that has been infused with information about the desired properties, without forgetting the general knowledge captured by the pretrained model. Development of such a framework should uncover parts of the protein design space that are unlikely to be found from either the generative model or predictive model alone. It is also desirable that such a framework can handle multiple properties, which might be in tension with one another, such as enzyme specificity and activity.

We propose ProteinGuide (Fig. 1) to fill this gap. Specifically, ProteinGuide enables on-the-fly statistical conditioning, meaning that it does not require training a new generative model; instead, ProteinGuide iteratively modulates or ‘guides’ the generation process of a pretrained model with information from one or more property-predictive models. Each sampling step in this guidance procedure represents a sampling step from the desired conditional generative model, without the need to retrain a generative model to explicitly capture the desired conditional distribution.

To demonstrate the potential use of ProteinGuide, we deployed it on a range of problems in protein engineering, first evaluating the approach *in silico* and then in the wet lab. Specifically, we demonstrate the applicability of ProteinGuide *in silico* by guiding different pretrained protein generative models, including ProteinMPNN, ESM3 and ESM-C¹⁶, to generate amino acid and structure token sequences, conditioned on several user-specified properties. These properties include higher stability and activity, as well as functional annotations such as EC (Enzyme Commission) numbers or CATH (class, architecture, topology, homology) classes. Our experiments span two important settings, namely generating sequences with similar properties to the training data (interpolative) and property values beyond those seen in the training data (extrapolative). We also demonstrate that ProteinGuide can be used on more challenging problems wherein we seek to design for two different desired properties that are in tension with one another (increased on-target and decreased off-target activity). Lastly, we apply ProteinGuide with two different inverse-folding models to design adenine base editors (ABEs) that we synthesized and assayed. Specifically, we obtained activity measurements for 2,000 variants sampled from the pretrained models, which we used to train a predictive model. We then used ProteinGuide guided by this predictive model to generate 2,000 more variants intended to have higher editing activity. We found that we can surpass the editing efficiency previously achieved with seven rounds of directed evolution, with just a single round of guided design with ProteinGuide.

Results

We begin by introducing the concept of guidance and the theoretical framework underlying ProteinGuide, from which we derive the guidance algorithms used throughout this work. We then place ProteinGuide in the context of alternative strategies for incorporating auxiliary information into generative models, before presenting our *in silico* and *in vivo* experimental results.

Intuitive overview of guidance

Conceptually, performing guidance comprises taking (1) a pretrained generative model, $p(x)$, on sequences x and (2) a classification or regression predictive model, $p(y|x)$ for property y , so as to enable sampling (that is, generation) from the desired conditional generative model,

$x - p(x|y) \propto p(y|x)p(x)$. Notably, guidance does so without training or retraining a generative model on x . Rather, it modulates the sampling procedure used on the pretrained model by executing the Bayes rule at each sampling step in an iterative fashion. At each step, a property-conditioned distribution for the next amino acid is computed using both the pretrained model and the predictive model. The upshot is that the resulting sampled sequence is a sample from the desired property-conditioned generative model, notably without ever having explicitly constructed the corresponding conditional generative model itself. Therefore, we refer to guidance as on-the-fly conditioning, as it does not require training a generative model; it only modulates the sampling process of the pretrained model.

We first develop guidance for discrete-space diffusion models (DDMs) and discrete-space flow-matching models (DFMs), a method we refer to herein as Discrete Guidance¹⁷, which extends guidance methods originally developed for continuous spaces^{18–21} to discrete spaces. In continuous spaces, diffusion models estimate the gradient of the log probability of the state, $\nabla_x \log p(x)$ —the score function—as the core object. The key insight enabling guidance is that each step of the sampling process, which iteratively uses the score function, can use the Bayes rule: $\nabla_x \log p(x|y) = \nabla_x \log p(x) + \nabla_x \log p(y|x)$. Consequently, the corresponding sequence of gradient steps enables sampling from the desired conditional distribution, $x - p(x|y)$ ^{18–20}. However, for DDMs and DFMs, these gradients are not defined and such an approach cannot be readily adopted¹⁷. In fact, in discrete spaces, such models need to be set up completely differently even without guidance, to rely on transition rates instead of gradients^{22–26}. The transition rates dictate the probability of an amino acid at a position being sampled at each step of the iterative sampling procedure. In an analogous manner to deriving conditional gradients for continuous-space diffusion guidance, we previously derived conditional transition rates to unlock guidance for diffusion and flow-matching models on discrete spaces such as biological sequences¹⁷ (Methods).

Our derivation therein showed that, to sample from the desired conditional generative model, one modulates the unguided transition rates with a likelihood ratio of the desired property condition before and after each possible transition (Methods). The key to enabling practical application of guidance with these conditional rates is to leverage properties of the continuous-time process to enable what is naively an intractably expensive computation to one that is manageable. Specifically, we showed how to change naive guidance from being combinatorially expensive in the sequence length at each sampling step, to only linear in sequence length, by noticing that it is sufficient to assume that only one position mutates in infinitesimal time (Methods). We also showed how to speed up this exact guidance even further by approximating repeated calls to the predictive model with a first-order approximation to the predictive model. This even faster, approximate guidance, called Taylor-approximate guidance (TAG), matched the sample quality of exact guidance on several tasks¹⁷.

Overview of ProteinGuide

ProteinGuide is a unifying framework that extends Discrete Guidance to a broad range of discrete-space generative models by exploiting equivalences in their training objectives and sampling distributions, shown herein (Methods). A key observation is that, when diffusion and flow-based models use a so-called ‘masking’ noise process—a common choice for protein sequence modeling—their training objectives and sampling distributions reduce to forms that are mathematically equivalent to those of any-order autoregressive models (AO-ARMs) and generative masked language models (MLMs) (Methods). As a result, four widely used model classes—DDMs, DFMs, MLMs and AO-ARMs—can be unified under a single framework comprising both the continuous-time models (DDMs and DFMs) and the discrete-time models (MLMs and AO-ARMs) (Fig. 1b). Herein, we establish the training and sampling equivalences of all four of these models, enabling ProteinGuide to

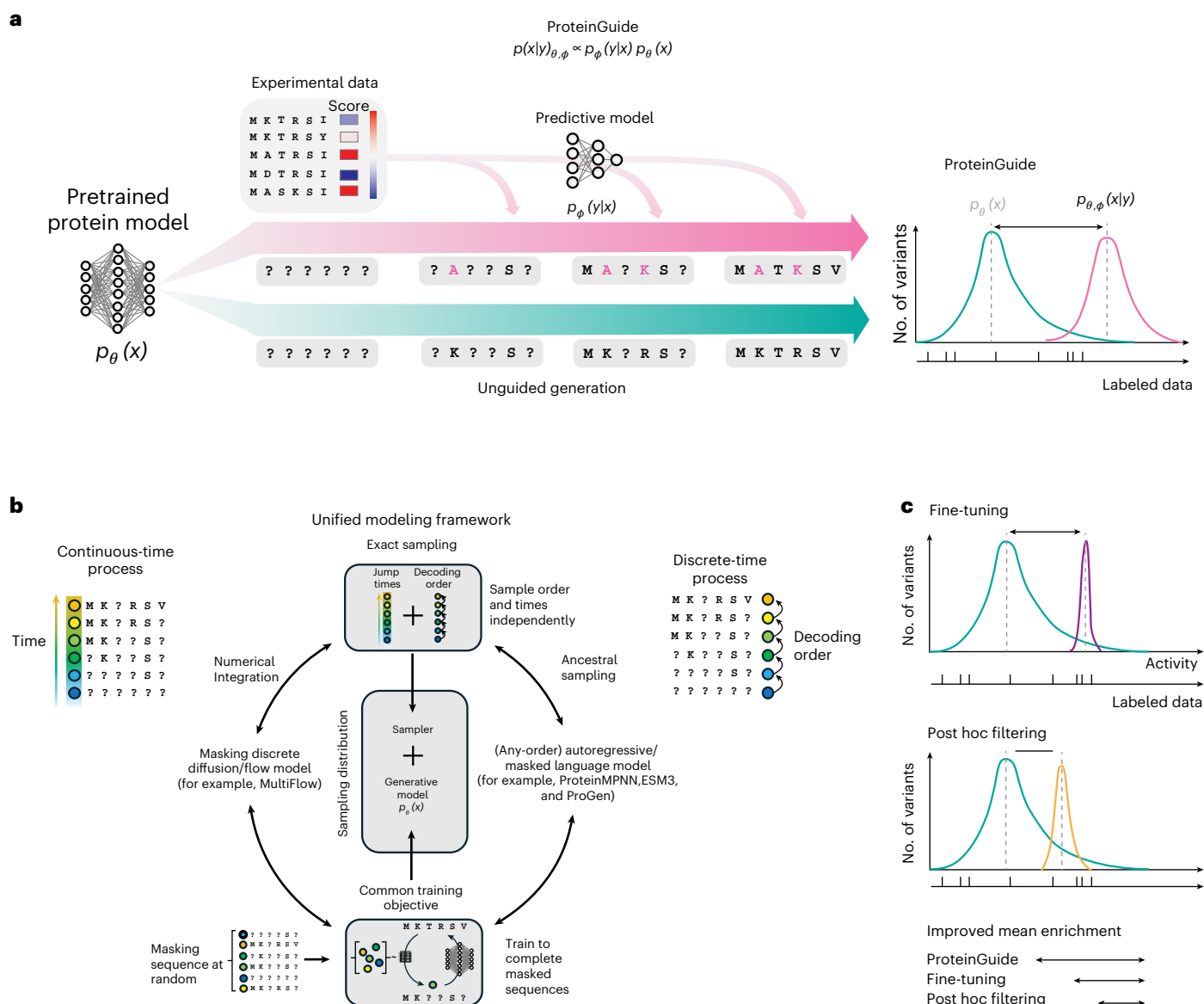


Fig. 1 | Overview of ProteinGuide. a, Contrasting unguided generation with guided generation by way of ProteinGuide. A sequence is denoted as x ; the pretrained generative model is denoted as $p_{\theta}(x)$; y denotes a property of interest that we seek to guide toward using predictive model $p_{\phi}(y|x)$. When guiding, we blend information from the predictive model with that of the pretrained model in a statistically correct manner, resulting in sequences sampled from the property-informed, reweighted distribution, $p_{\theta,\phi}(x|y)$. Masking of one sequence position is denoted with the symbol “?” and pink letters indicate key substitutions that are introduced only when guiding with the predictive model, rather than when using unguided generation. **b**, Unifying view of training and sampling for masked diffusion or flow models, MLMs and AO-ARMs enables guidance on a broad set of model classes. Colored dots visually tag individual sequences, each of which

has a different amount of masked tokens. The sequence being masked during training is MKTRSV. The shared training objective involves randomly masking out parts of the sequence and then predicting the masked sequence given the remaining unmasked context over different masking rates. Once any type of model is trained, one can then use a sampling algorithm originally developed for any of the other models. Consequently, we can obtain computationally tractable exact sampling for continuous-time models and also accelerated approximate sampling. **c**, Schematic contrasting ProteinGuide with post hoc filtering and fine-tuning. ProteinGuide combines a pretrained generative prior with assay-labeled data through correct statistical conditioning, enabling more reliable enrichment for high-property sequences than the other approaches shown.

function as a generalization of Discrete Guidance to all these model classes. Thus, any model belonging to one of those four classes can be guided. Moreover, our unifying view also allows us to correctly sample from the continuous-time models (with masking noise processes) as though they were discrete-time models (AO-ARM-style) and vice versa.

Although sampling as though the model were one in discrete time is equivalent to sampling it as though it were in continuous time, each viewpoint can lead to different practical algorithms each with its own pros and cons. Next, we mention two sampling algorithms used herein. First, when viewed from the continuous-time perspective, we

can make use of our first-order TAG¹⁷, useful because it is fast despite being approximate. We used TAG in the majority of our experiments herein. The second is our derived method that we call discrete-time exact guidance (DEG). This method samples from the desired conditional distribution exactly but is slower than TAG when the predictive model is too computationally expensive to evaluate all substitutions at a given position in a single batch. Consequently, it is advisable to use DEG when numerical integration with TAG is unstable, when TAG’s first-order approximation of the likelihood ratio is poor or when the predictive model can be evaluated quickly enough. This was the case

in both our multiproperty Pareto extrapolative and enzyme class guidance experiments, where we, thus, used DEG.

While connections among the training objectives of the four model classes have been previously noted^{27–31}, the practical utility of these observations has been unclear until now. Moreover, the formal bidirectional connection between discrete-time and continuous-time exact sampling procedures has not, to the best of our knowledge, been previously discussed (Methods). Our unifying view of training and sampling brings these four model classes under a single framework, enabling ProteinGuide to be applied to a broad set of discrete-space generative models, and is, thus, applicable to many popular protein sequence generative models used by protein engineers (Fig. 1b).

Alternative approaches for infusing generative models with auxiliary information

The simplest approach for generating sequences from pretrained models with desired property criteria is to post hoc filter the generated sequences to only those satisfying the user-specified criteria (Fig. 1c). Such an approach is practical when the pretrained model by itself can generate sequences satisfying the criteria with reasonable frequency. However, more generally, such an approach is inefficient to the extent that the properties of interest are not well represented in the pretrained model, as we see in our experiments below. Additionally, for two uncorrelated properties one wishes to satisfy, the frequency argument becomes worse still, let alone two properties in tension with each other (anticorrelated). Any approach that stands a chance of outperforming post hoc filtering must involve reshaping the probability mass of the generative model with knowledge of the sequence–property relationships of interest. This is what ProteinGuide does.

Perhaps the most commonly used approach in protein engineering for infusing experimental data into a generative model is by way of fine-tuning^{8,9}. With fine-tuning, the pretrained generative model is trained further, with its original objective, on a small, curated set of examples that have the desired properties (Fig. 1c); this procedure is also sometimes referred to as supervised fine-tuning in the protein engineering community, which we forgo because no label is used during training. The curated fine-tuning set might comprise sequences from a particular family that one hopes to focus the model on^{10,11,13,32,33} or sequences that are deemed functional for a property of interest^{34,35}. Fine-tuning has the benefit of being immediately applicable to any class of generative models. However, it has two downsides. First, fine-tuning is a heuristic method for combining sources of information for which the statistical outcome is not well defined, as opposed to, say, ProteinGuide, which executes the Bayes rule. Consequently, important information from the pretrained model may be lost, a phenomenon often referred to as ‘forgetting’ (refs. 14,15). Second, the act of continuing to train with the original objective on new data concentrates the existing pretrained generative model distribution around the fine-tuning data, limiting generation to sequences whose properties resemble those seen during training^{10,11,13}, which may be useful for interpolative design tasks but less so for extrapolative tasks. We see evidence of this struggle to extrapolate in several of our *in silico* experiments below. Nevertheless, fine-tuning has been shown to be useful either on its own or as a first step before applying other approaches^{8,9,34} and is likely sufficient for designing proteins with similar property values to those seen in the training data.

Fundamentally, guidance and, consequently, ProteinGuide differ from both post hoc filtering and fine-tuning in that they use a standalone predictive model to infuse auxiliary information into the pretrained model using the statistical operation of the Bayes rule. Consequently, ProteinGuide provides a framework with which to understand how we are blending sources of information. Moreover, for the same reason, it can coherently blend several predictive models to simultaneously condition on multiple properties, as shown later. We caution that ProteinGuide can only be as good as the information in its

two underlying component models, not least of which is the predictive model and its ability to extrapolate if desired.

Other statistically grounded frameworks have also been proposed. For example, direct preference optimization (DPO)³⁶ and related work^{37–40} enable Bayesian-style updating of a pretrained model using ‘preference-labeled’ data or a user-specified ‘reward function’. Notably, all of these methods require training a new generative model before one can sample from it. Additionally, DPO is applicable only to models with tractable likelihoods, thereby largely restricting it to single-order ARMs³⁷; however, it has recently been extended to diffusion models on continuous spaces by way of approximation⁴¹. Further research is needed to determine whether DPO might similarly be extended to the discrete-space models considered herein, for which exact likelihoods are also intractable (Supplementary Text). Herein, we focus our study on on-the-fly guidance, rather than approaches such as DPO that require training a generative model. However, we believe that approaches that are not on the fly will also prove useful.

Guidance offers an approach to statistical conditioning that is theoretically sound, practically well behaved, flexible and complementary to existing techniques. Importantly, guidance does not require access to the parameters of the pretrained model; it only requires predictions (logits) from the model. Accordingly, it can be used with closed-source models and large open-source models for which fine-tuning might not be feasible for many users. Additionally, in settings where useful property-predictive models already exist, guidance allows one to directly leverage those models for the design task at hand. In recent work comparing our earlier guidance-based approach to fine-tuning for protein property-focused design tasks, it was found that Discrete Guidance¹⁷ performed favorably compared to fine-tuning based approaches and that guidance required less hyperparameter tuning and computation, making it more practically accessible to everyday users than other approaches examined⁴². A further discussion of related work, including that of Markov chain Monte Carlo (MCMC) methods^{43–46}, is provided in the Supplementary Text.

Overview of *in silico* experiments

To illustrate the potential of ProteinGuide, we first applied it, *in silico*, to a suite of experiments spanning three design settings, using a set of representative, well-known protein generative models, including ProteinMPNN¹, ESM3 (ref. 47) and ESM C¹⁶, as illustrative examples. An overview of these experiments is presented in Extended Data Table 1. Note that ProteinGuide can also be applied to many other sequence generative models^{11,25,48–51}.

The three design settings capture three valuable use cases. The first is an interpolative design setting, wherein we sought to generate proteins with property values in the range seen in the supervised training data but with greater diversity than appeared there. In the second category of extrapolative design, we sought to generate diverse proteins with property values that exceeded those seen in the supervised training data. Lastly, in the third category, multiproperty, Pareto extrapolative, we sought to increase one property while decreasing another, such that we could push beyond their optimality frontier delineated by the supervised training data.

For interpolative tasks, we guided the inverse-folding model ProteinMPNN with a stability predictive model to generate sequences; we also guided the multimodal model ESM3 with function annotations to generate sequences and structures. Because the interpolative setting is a relatively easy one for design, we highlight the former in the main text, leaving the other to the Methods (Extended Data Figs. 1 and 2). For the extrapolative tasks, we guided ESM3 to generate sequences with increased fluorescence, ubiquitin ligase activity or *in vivo* functional complementation. We focused on one protein for each property, using a predictive model trained with relevant assay-labeled data. For the extrapolative, multiproperty Pareto task, we guided ESM C to generate PbrR sequences for two properties simultaneously, seeking to

extend each one beyond the range in the training data. Specifically, we sought sequences that bind the desired lead ion with higher affinity and bind the off-target Zn ion with lower affinity. The interpolative tasks had hundreds of thousands of variants with which to train the predictive model, while the extrapolative tasks had around 2,000 and the Pareto extrapolative task had around 1,000. We chose 2,000 for the extrapolative task so that it would match our assay capacity for our own extrapolative wet-lab design experiments, whereas the other numbers were defined by the datasets used.

In all of these *in silico* experiments we compared ProteinGuide to the two most commonly used approaches in practice: post hoc filtering ('unguided, filtered') and fine-tuning. When fine-tuning ESM3 and ESM C, we used LoRA⁵². Methods were evaluated on the k best samples (best according to the 'oracle' model) of a larger set of samples. The size of the larger set was controlled by keeping the wall-clock time constant across all methods and generating as many as possible. In particular, we used the wall-clock time needed for ProteinGuide to generate k samples. We used $k = 100$ for all *in silico* experiments in the main text. To gauge the difficulty of the design task we had set for ourselves, we also compared to use of the pretrained model by itself (unguided), letting it generate k samples so as to compare directly to ProteinGuide on a per-sample basis. The extent to which the pretrained model by itself performs poorly in this setting reflects that the chosen task is challenging. Note that the cost of making calls to the predictive model relative to the cost of sampling from the pretrained model changes the number of samples the other methods are allowed to use as it is based on the wall-clock time of ProteinGuide drawing k samples.

To perform *in silico* evaluation, we made use of an oracle model trained on the full assay-labeled dataset as a proxy to measuring the designed proteins in a wet lab. Such an evaluation protocol follows on the work of others^{34,42,43}. We distinguish between this model and the one used to guide with ProteinGuide by calling the former the oracle model and the latter the predictive model. For the interpolative experiments, the predictive model used to guide was trained on the same data as the oracle¹⁷; for the two types of extrapolative experiments, we disallowed the predictive model to have access to top-performing variants, whereas the oracle had access to all variants. For the Pareto extrapolative experiments, top-performing meant those variants lying on the Pareto frontier. Further details are provided below. Note that, unless otherwise noted, the oracle and predictive models were identical in architecture and training procedure, differing only in their training data.

Although, by necessity, the field has come to often depend on evaluation by way of such oracle models, the manner in which the oracle differs from the predictive model may not fully represent real use cases. Consequently, wet-lab validation rounds out the evaluation. However, wet-lab validation, by itself, is also limited because of the limited scope of experiments that can be performed because of cost, time and labor. Performing both systematic *in silico* and wet-lab experiments provides the best basis on which to make conclusions.

In addition to metrics based on the oracle, we evaluated the diversity and novelty of the generated sequences. Diversity was computed as the averaged pairwise Hamming distance among the generated sequences and novelty was computed as the minimum Hamming distance of generated sequences to any sequence in the full labeled dataset. Full experimental details, including datasets, predictive models, baseline configurations, sampling settings and additional results are provided in the Methods and Supplementary Information. Next, we go through the *in silico* experimental results.

Interpolative experiments: guiding ProteinMPNN for improved stability

Here, we illustrate how assay-labeled experimental data can be used to condition protein sequence generative models to improve their success in generating sequences with a desired function. We used ProteinGuide

to guide ProteinMPNN (an AO-ARM) with a predictive model trained on a large dataset of experimental folding stability measurements of miniproteins⁵³. The goal was to generate sequences with greater stability than the wild type (Fig. 2a). A similar experiment was previously conducted by retraining an inverse-folding model with a variant of DPO³⁷; ProteinGuide lets us bypass retraining an inverse-folding model.

We conducted these experiments on eight randomly chosen proteins from the stability dataset for which sequences generated by ProteinMPNN were less stable than the wild-type sequence on average. For each of these proteins, the corresponding stability dataset contains 1,000–3,000 single and double mutants. The success rate for each protein was computed over $k = 100$ generated sequences, where success was defined on the basis of two criteria: (1) the generated sequence should fold into the desired structure and (2) the generated sequence should be at least as stable as the wild-type sequence, $\Delta\Delta G \leq 0$. We used ESM3 to predict the structure of the generated sequences and then computed the root-mean-square deviation (r.m.s.d.) between the predicted and desired structures. We defined structure success as $\text{r.m.s.d.} \leq 2 \text{ \AA}$, following previously used evaluation criteria²⁵. Folding stability was assessed with the stability oracle model from Nisonoff et al.¹⁷. For the fine-tuning dataset, we used all sequences more stable than the corresponding wild type, similar to Widatalla et al.³⁷.

Across all eight proteins, ProteinGuide consistently yielded the highest success rates (Fig. 2b). In particular, the generated sequences from ProteinGuide preserved the structure constraints of the backbone (Fig. 2c), while simultaneously improving predicted stability (Fig. 2d). As expected, post hoc filtering had most success when the unguided model itself had at least a moderate success rate. Fine-tuning improved success rates of the unguided model in some cases but failed to do so in others, for reasons that are not yet fully understood. In contrast to the baselines, ProteinGuide worked well across the board. Note that diversity and novelty of generated sequences was similar across all methods (Extended Data Fig. 3). In these experiments, we sampled sequences in the same manner as in Nisonoff et al.¹⁷ (Methods). However, we also ran the same comparisons using autoregressive-based sampling, for which the results were qualitatively similar (Supplementary Fig. 1).

Extrapolative experiments: guiding ESM3 to generate sequences from a family for desired properties

Having investigated the interpolative design setting, we next evaluated ProteinGuide with the more challenging task of generating sequences with higher property values than observed in the assay-labeled datasets (Fig. 3a). We focused on three different proteins, each with a different assay measuring a different property: (1) the CreiLOV fluorescent protein, where the assay measured fluorescence intensity⁵⁴; (2) the ubiquitination factor E4B (UBE4B), where the assay measured E3 ubiquitin ligase activity⁵⁵; and (3) the poly(A)-binding protein Pab1 (PABP), where activity was measured by way of an *in vivo* functional complementation selection⁵⁶. For the three datasets, the mean numbers of substitutions away from the wild type were eight, two and two, respectively (Extended Data Fig. 4a).

For our pretrained model, we used ESM3 as an unconditional amino acid sequence generative model. For each of the three proteins, we guided generation with a predictive model trained on 2,000 randomly chosen labeled sequences from that protein family. In the previous section, we guided an inverse-folding model conditioned on the structure of a wild type for one family; hence, the pretrained model was already effectively concentrated on a family. Here, we do not use ESM3 with any conditioning; thus, as a pretrained model, it will sample from all families. Consequently, in addition to guiding with our property-predictive model, we simultaneously guide ESM3 toward the family of interest. To do so, we trained a predictive model to discriminate between sequences in the assay-labeled dataset and sequences generated from pretrained (unguided) ESM3 (Methods); we treated the resulting predictive model as predicting whether a variant belonged to

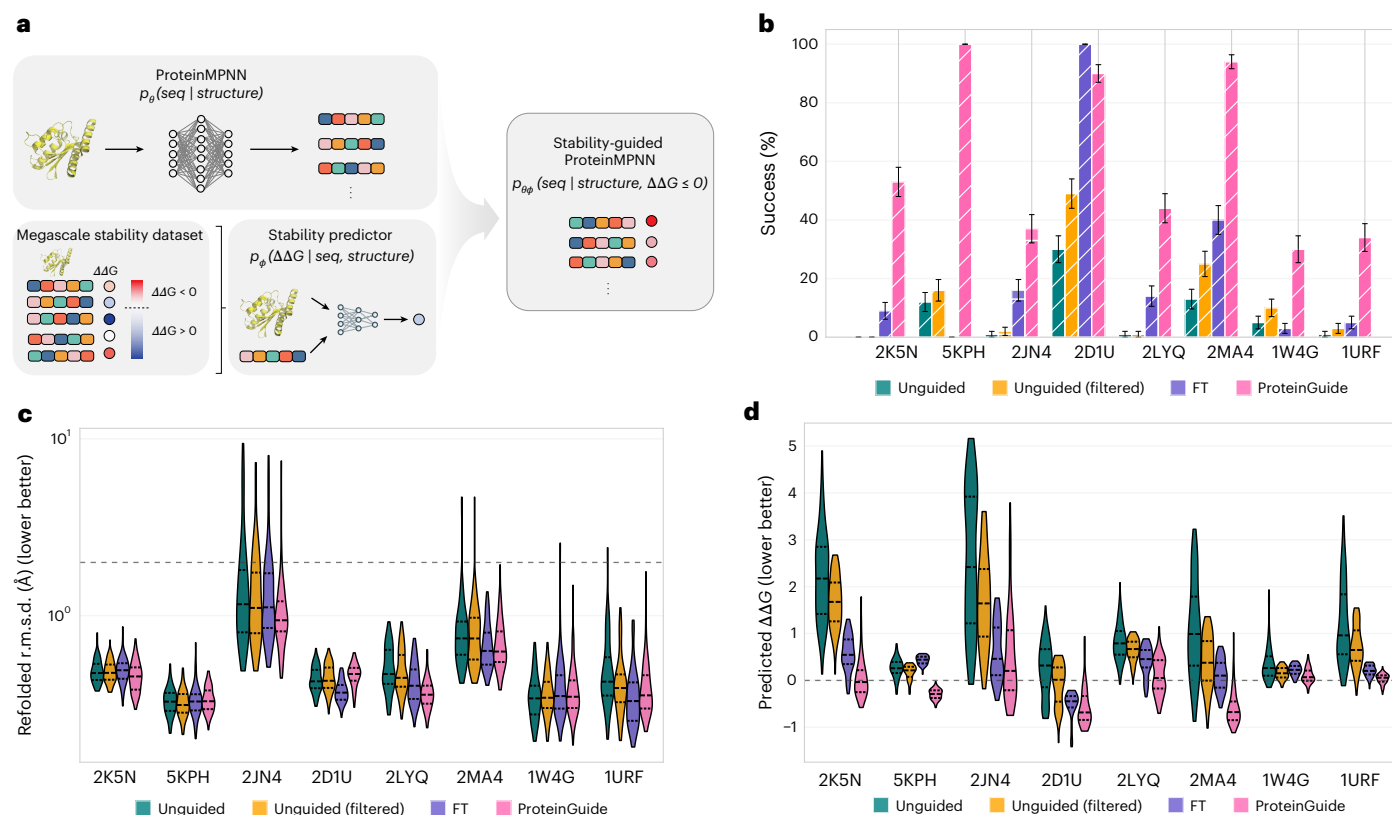


Fig. 2 | Guiding ProteinMPNN with experimental stability data to generate sequences with enhanced stability. **a**, Schematics for guiding ProteinMPNN with large-scale experimental stability measurements. ProteinMPNN, an inverse-folding model, is guided by a stability predictive model trained on the large-scale stability dataset from Tsuboyama et al.⁵³ to generate sequences with stability better than the wild-type sequence. **b**, Four methods for sampling (in legends) are assessed across eight proteins (horizontal axis) by their success rate on $k = 100$ generated samples. The height of each bar represents the percentage of sequences predicted to be at least as stable as the wild type ($\Delta\Delta G \leq 0$), equivalent to the empirical mean of a binary success indicator, while the hatched subportion indicates the fraction of those stable sequences that are

also predicted to correctly fold into the desired structure (r.m.s.d. ≤ 2 Å). In most cases, the hatched subportion covers the full height of the bar. Error bars represent ± 1 s.e.m., calculated as $\sqrt{\hat{p}(1-\hat{p})/k}$, where $\hat{p}$ is the empirical proportion of success and $k = 100$ is the sample size. **c**, Distributions of refolded r.m.s.d. for the generated sequences (lower is better). The dashed horizontal line indicates the threshold below which a sequence is considered to fold into the desired structure (r.m.s.d. ≤ 2 Å). **d**, Distributions of oracle-predicted stability for the generated sequences (lower is better). The dashed horizontal line indicates the threshold below which a sequence is predicted to be at least as stable as the wild type ($\Delta\Delta G \leq 0$).

the protein family of the assay-labeled data's wild type. For fine-tuning, we trained on the top M percentile of variants from the assay-labeled dataset, where M was chosen by scanning through different percentile thresholds ($M = 2, 5, 10, 20, 50$ and 100) on top-scoring variants as a hyperparameter and choosing the one that had highest likelihood for held-out high-scoring variants. $M = 5$ was selected for all three datasets.

For each protein design task and method, $k = 100$ sequences were used to compute the percentage of these variants that were successful; success was defined as a variant (1) having an oracle-predicted property above a predefined percentile, τ , in the labeled dataset and (2) not having been observed in the labeled dataset. We used each of these percentile thresholds in turn, $\tau = 20, 50, 80$ and 100 . We withheld top-performing variants when training each predictive model that was used to guide, whereas the oracle model had access to all variants (100–200 variants withheld; Methods), so as to mimic the fact that, in a real setting, the predictive model may not accurately extrapolate.

Across all three protein design tasks, ProteinGuide consistently yielded the highest success rates. In particular, ProteinGuide generated sequences with highest oracle property values compared to other methods (Fig. 3b) and with higher diversity and novelty than those generated with fine-tuning (Fig. 3c). Moreover, some of ProteinGuide's variants had higher property values than any observed in the assay-labeled dataset (Fig. 3d). As expected, fine-tuning samples' oracle values recapitulated the range of those in its training dataset

(Fig. 3c,d). We hypothesize that this occurred because fine-tuning tends to concentrate the distribution of the pretrained model on the new training data—most useful for interpolative design. ProteinGuide, on the other hand, has the potential to extrapolate well to the extent that the predictive model used for guidance is able to extrapolate sufficiently accurately beyond its training data.

Multiproperty, Pareto extrapolative experiment: guiding ESM C for on-target and off-target binding of PbrR

In our last series of in silico experiments, we tackled a design problem that required optimizing for two properties in tension with one another. This setting is of great interest to protein engineers who may want to, say, increase the activity of an enzyme while simultaneously keeping its stability above some threshold (because increasing activity may decrease stability). Moreover, practitioners may want to turn a 'knob' to trace out the optimality (that is, Pareto) frontier to characterize it. More generally, one may seek to optimize simultaneously for more than two properties, which in principle is also possible using ProteinGuide.

Our multiproperty optimization task uses data from a recent design campaign that optimized the PbrR protein for use in a cell-free biosensor to detect lead (Pb) contamination in water⁵⁷. PbrR is a transcription factor that forms a dimer when bound to metal ions—typically Pb(II)—allowing it to bind DNA and induce the transcription of a downstream reporter gene. The key challenge is that increasing Pb affinity

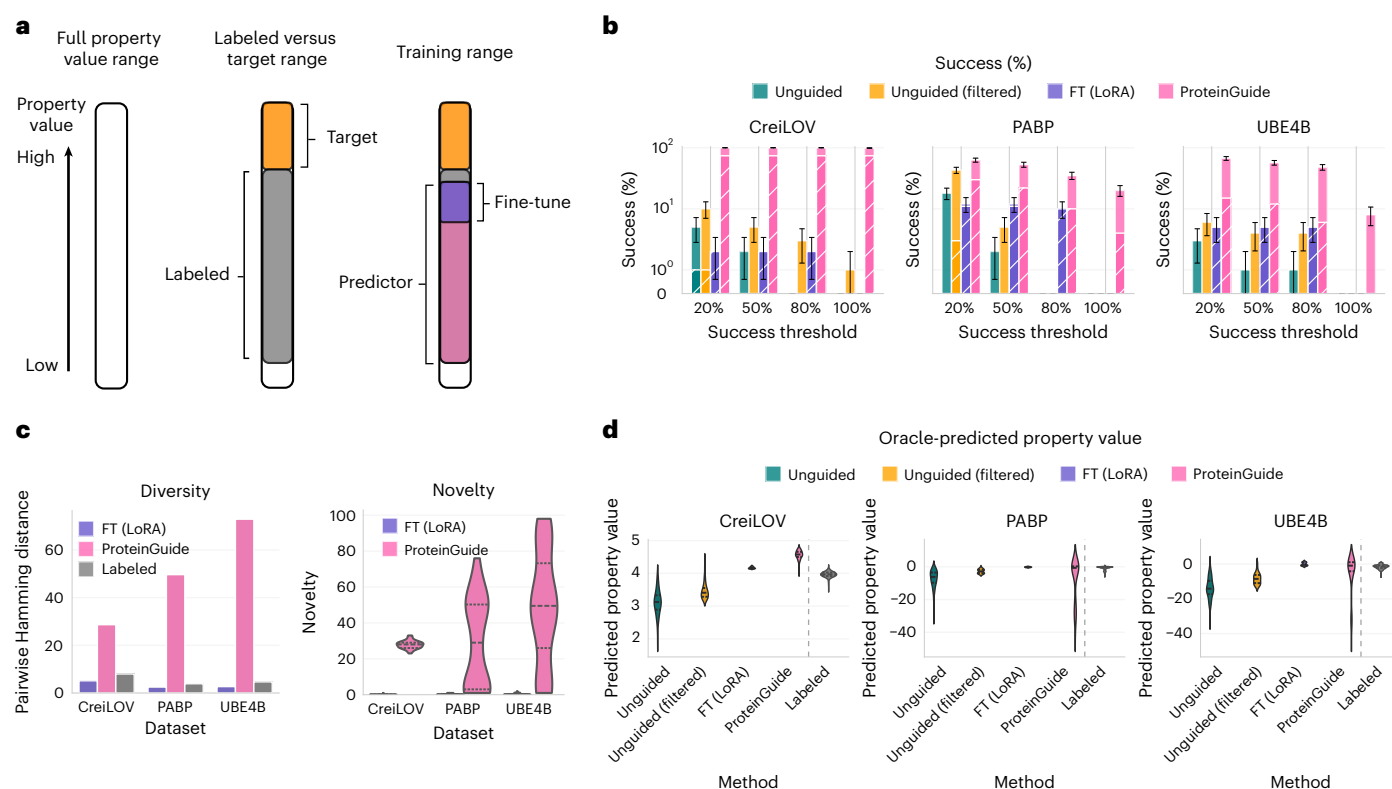


Fig. 3 | ProteinGuide guides ESM3 with assay-informed predictive models for extrapolative design. a, Experimental setup for the extrapolative setting. **b**, For each dataset, each method is scored by its success rate on $k = 100$ samples. The height of each bar represents the percentage of unique sequences predicted by the oracle model to be above a particular threshold based on the percentile of the full labeled dataset (horizontal axis) and are not identical to any sequence in the labeled set (that is, novelty > 0), while the hatched subportion indicates the fraction of those sequences that are also predicted by ESMFold⁷¹ to fold into a structure with pLDDT ≥ 0.8 (Supplementary Fig. 2). Error bars represent ± 1 s.e.m., calculated as $\sqrt{\hat{p}(1-\hat{p})/k}$, where $\hat{p}$ is the empirical proportion of success and

$k = 100$ is the sample size. FT, fine-tuning. **c**, Diversity (left) and novelty (right) of the generated sequences. Higher diversity and novelty are desirable provided that the sequences also have high property value. Diversity is computed as the averaged pairwise Hamming distance among the generated sequences. Labeled diversity is computed on 500 randomly sampled sequences from the full labeled dataset for each protein. For each sequence, novelty is computed as its minimum Hamming distance with sequences in the full labeled dataset. The novelty of fine-tuning generated sequences is very low and is not readily discernible. **d**, Distributions of oracle predicted property value for the generated sequences. 'Labeled' consists of all sequences in the full labeled dataset.

typically also increases off-target binding to zinc (Zn). Therefore, we aimed to generate PbrR variants to simultaneously increase Pb binding while reducing off-target Zn(II) binding (Fig. 4a). Accordingly, the goal was to push the Pareto frontier of the experimentally collected data outward (that is, past the empirically observed limits of joint optimality for this system; Fig. 4d, blue line).

To simulate the early stage of a design campaign, we only used experimental data from the first library (of five) collected by Wang et al.⁵⁷. This first library comprised 1,098 variants arising from an alanine scan of the protein, a site saturation mutagenesis (SSM) at 49 positions and a modest number of higher-order substitutions (up to a degree of 5) sampled from seven positions. We only designed positions that had SSM data available and, hence, could have meaningful predictions from the model. The remaining positions were fixed as their wild-type amino acids. Binding affinities were reported as log fold change (log FC) of fluorescence with respect to the wild-type protein in the presence of its metal-ion cofactor.

In this experiment, we also sought to mimic the common scenario in protein design where initial screens may not capture higher-order epistatic interactions, resulting in the fact that a predictive model trained on such data cannot capture such higher-order information. Consequently, in this experiment, the oracle model was allowed to capture up to third-order interactions, whereas the predictive model used for guidance was restricted to learn only up to second-order interactions. Thus, we trained two separate second-order regression

models for guidance, one for each of Pb-binding and Zn-binding affinities; similarly, we trained two corresponding third-order oracle models. We used ESMC¹⁶ as the pretrained model. Fine-tuning was performed with the variants lying on the experimentally observed Pareto frontier for on-target and off-target activity (Extended Data Fig. 5a, middle).

We evaluated our methods on two different design tasks. In the first task—Pareto extrapolation—we aimed to generate sequences in a target region that lay outside the (approximate) Pareto frontier of the experimental data. Specifically, we sought to produce sequences with Pb log FC > 2.0 and Zn log FC < -0.6 (Fig. 4b, yellow rectangle). In the second task—controllable generation—we selected three reference points along the Pareto frontier representing different Pb-binding and Zn-binding trade-offs (Fig. 4d, leftmost plot, points on the blue line). For each reference point, the target region for design that we aimed for corresponded to the part of function space with strictly better Pb-binding and Zn-binding characteristics than the reference point (Fig. 4d, leftmost plot, quadrants delineated by colored dashed lines).

For the Pareto extrapolation task, 75% of the sequences generated using ProteinGuide fell into the design target region (Fig. 4b), without losing any diversity compared to sequences generated with post hoc filtering (Fig. 4c). For post hoc filtering, only 16% of the sequences fell into the target region. The fine-tuned model's samples remained very close to the fine-tuning training data and failed to extrapolate beyond the original data's Pareto frontier (Fig. 4b). In fact, fine-tuning the pretrained model reduces (that is, worsens) the log likelihood of the

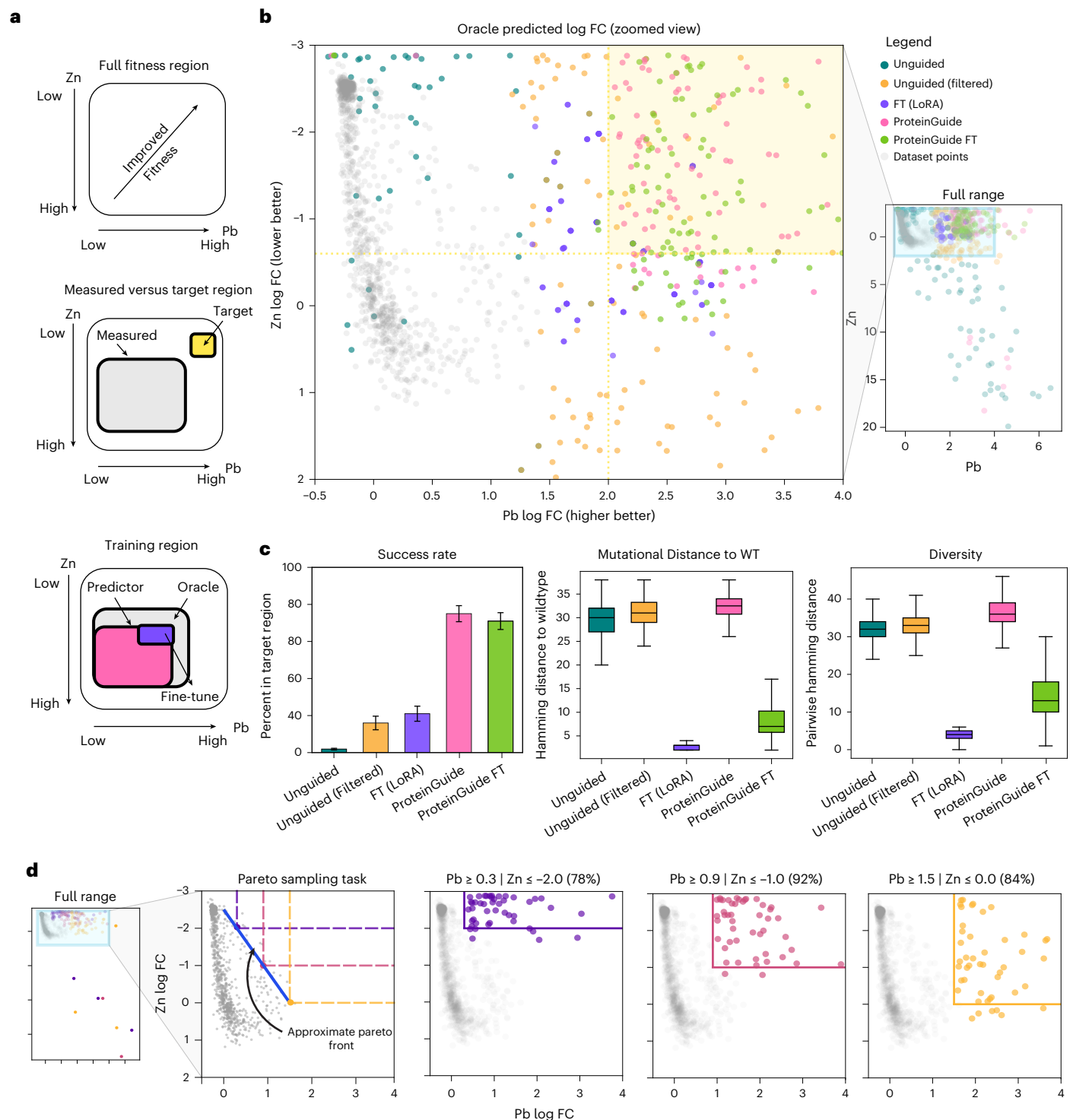


Fig. 4 | Multiproperty guidance with ProteinGuide on PbrR to improve Pb binding and reduce off-target Zn binding. **a**, Schematic diagrams of the data splits used for training and evaluating models for multiproperty guidance on PbrR. **b**, Pareto extrapolation task. Scatter plot of oracle-predicted log FC for model generations for ESM C (unguided and unguided (filtered)), fine-tuning, ProteinGuide and ProteinGuide fine-tuning (FT). Experimental values of the initial library of variants from Wang et al.⁵⁷ are also shown in gray. The target region requires a Pb log FC of 2.0 and a Zn log FC of -0.6 , where both axes are normalized such that the wild-type log FC is 1.0. **c**, Left, the proportion of sequences that were in the target region for each of ESM C (unguided and unguided (filtered)), fine-tuning, ProteinGuide and ProteinGuide fine-tuning.

Middle, the Hamming distance between sequences generated by each method and the wild-type sequence. Right, the average Hamming distance between sequences generated by each method, as a measure of diversity. WT, wild type. **d**, Controllable generation task. The leftmost plot highlights outliers and the region corresponding to the remaining four plots (in yellow shading). The second plot from the left shows each of three target regions for the Pareto sampling task as quadrants delineated by colored dashed lines. Each target region has its bottom left vertex anchored on the approximate Pareto frontier. The rightmost three plots show sequences generated by guidance for each target region using the classifier thresholds in the subpanel titles. The success rates are also shown in parentheses.

sequences in the target region compared to the pretrained model itself (Extended Data Fig. 6f). Additionally, sequences generated from the fine-tuned model displayed poor diversity (Fig. 4c).

In the previous section, we guided ESM3 with a predictive model of whether the sequence belonged to the relevant family, in addition to the property-predictive model. For this experiment, we additionally included ProteinGuided fine-tuning, which is an alternative approach for achieving a similar end. Instead of guiding with a second ‘protein family’ model, ProteinGuided fine-tuning involves first fine-tuning the pretrained model (ESM C) to focus it on the family of interest and then applying ProteinGuide on top of it. Specifically, we used the same fine-tuning model described in our experiments just above as the pretrained model for ProteinGuide.

We found that there was no significant difference in the number of sequences that fall into the target region between ProteinGuided fine-tuning and ProteinGuide alone (Fig. 4c). However, the potential advantage of ProteinGuided fine-tuning is that the fine-tuned model on which we apply ProteinGuide is already in the ballpark of where we expect to most trust the predictive model: concentrated on sequences with a small mutational distance to the wild type, like the predictive model’s training data. ProteinGuided fine-tuning makes 20 fewer substitutions on average compared to ProteinGuide (Fig. 4c, middle). Because samples drawn from the guided, fine-tuned model have predicted property values similar to those drawn from the guided, pretrained model alone, our results suggest that the approach of fine-tuning first may be promising for generating samples that extrapolate well.

In our next experiment with these same data, we wanted to showcase the ability of ProteinGuide to be easily redeployed to different but related engineering tasks. This could be useful, for example, over the course of several design–build–test–learn (DBTL) cycles, wherein one may want to optimize for pushing past the Pareto frontier according to different property trade-offs during different rounds of design. Thus, we ran three more design challenges, each with its own target region anchored on one point on the Pareto frontier (Fig. 4d). These anchor points represent different possible lower-bounds on the desired on-target or off-target binding affinities for the generated sequences that could be specified by a user during a DBTL cycle. Because ProteinGuide does conditioning on the fly, we need not retrain a new generative model for each of these related tasks. As our main goal was to showcase this capability, we did not compare to other models here. Overall, ProteinGuide achieved an average success rate of 85% across these three regions, showing that it could readily handle a diverse set of user-specified design objectives (Fig. 4d). Note that we evaluated on $k = 50$ samples in this experiment for expediency.

In vivo experiments: engineering a base editor with enhanced activity

At last, we move to an in vivo demonstration of ProteinGuide to engineer an ABE⁵⁸ for high editing activity (Fig. 5a). Base editors are engineered CRISPR effectors that can make transition mutations in the DNA by directly deaminating cytosine (causing C-to-T mutations) or adenine (A-to-G mutations) in the single-stranded DNA exposed during target recognition by catalytically inactivated Cas9. ABEs can fix about half of the pathogenic point mutations leading to genetic diseases⁵⁸. They have been evolved from an *Escherichia coli* tRNA-deaminase TadA through multiple rounds of directed evolution^{59–61}. ABEs are compact single-domain proteins (167 aa) and it has been shown that they can be evolved through at least two different evolutionary trajectories, leading to high DNA deamination activity^{59,61,62}. Furthermore, both evolutionary paths have led to high concentrations of mutations impacting the C-terminal region, suggesting that this region can be a promising target small enough to be chemically synthesized as an oligo pool while remaining functionally important for evolving DNA base-editing activity.

Our goal was to use experimental base editor activity data with ProteinGuide to condition pretrained generative models to produce sequences with improved activity above and beyond the activity of proteins that the models had access to. For pretrained models, we used two inverse-folding models, ESM3 and FMIF (flow-matching inverse folding)¹⁷—the latter being an inverse-folding model that we trained with similar training data and architecture as ProteinMPNN but having specifically removed all examples of engineered ABE variants from the training data. Our intent was to ensure that FMIF could not encode existing published information for engineering ABEs that is generally not available when engineering a new protein. We started our engineering campaign with the wild-type TadA sequence (an ABE), which is known to have very low, but detectable activity⁶³. Then, we chose to only design the region of 86 residues at the C-terminal region of this variant because of constraints of gene synthesis; we also hypothesized that this region might be enriched for mutations important for increasing editing activity according to prior studies^{59,61,62}.

Activity levels for the base editors were assayed using an in-house bacterial selection assay based on the restoration of an inactive antibiotic resistance marker to measure the activity of the library and activity was reported as log enrichments between the selective and nonselective conditions. Our first-round library—whose purpose was to obtain data to build a predictive model for guidance—was designed by sampling from each of the pretrained models, ESM3 and FMIF, separately (1,000 samples per model). These sequences were then pooled into a library of 2,000 variants that was then assayed for activity. From this dataset, we then built one classification model to be used for guidance, where the classifier predicts the probability that a sequence’s log enrichment is greater than zero (Extended Data Fig. 7).

Using this classifier, we next applied ProteinGuide to design an improved second-round library by guiding each of ESM3 and FMIF with it. Specifically, we generated 2,000 sequences by guiding each of the two pretrained models for 1,000 samples; we then assayed these 2,000 sequences for activity. These generated sequences were predicted to be more active on average compared to the first-round library according to the classifier, while maintaining a distribution of mutational distances to the wild-type TadA sequence similar to that of the first-round library (Fig. 5b).

We found that the majority of the ProteinGuide variants in the second round had higher activities (mean log enrichment: 0.66 ± 0.02) than the first-round library (mean log enrichment -0.28 ± 0.01) (Fig. 5c). Furthermore, the second-round library revealed variants that were more active than any seen in the first-round library, while having <60% sequence identity to the wild-type TadA sequence within the designed region (Fig. 5d). Intriguingly, we observed that, while in the first round, more variants from ESM3 were predicted to be active than from FMIF (Fig. 5b), the most active variants from the second round mostly came from guided FMIF rather than guided ESM3 (Fig. 5d). Moreover, when examining how amino acid preferences shifted from the pretrained model to the guided model (assessed by position-specific enrichment scores), for each of ESM3 and FMIF, we found that they were shifted differently, suggesting that ESM3 and FMIF may be capturing different effects (Fig. 5e). Note that the D108N substitution—enriched in sequences generated by the guided FMIF model—has been reported in prior directed evolution campaigns, whereas most other beneficial substitutions identified here differ from those previously observed (Fig. 5e)^{59,61}.

We chose two of the variants with high enrichment values and low sequence similarity to the wild type from our guided FMIF library for further validation (‘FMIF-1016’ and ‘FMIF-1041’; Fig. 5d, left). These sequences differed by at least 25% from any sequence in the first-round library within the designed region. Specifically, we tested their activity (1) individually in a titer plate-based assay and (2) in a pooled selection assay. In both assays, we also compared to the previously engineered base editors: ABE1.1, ABE7.10 and ABE8 (Methods)^{59,61}. According to

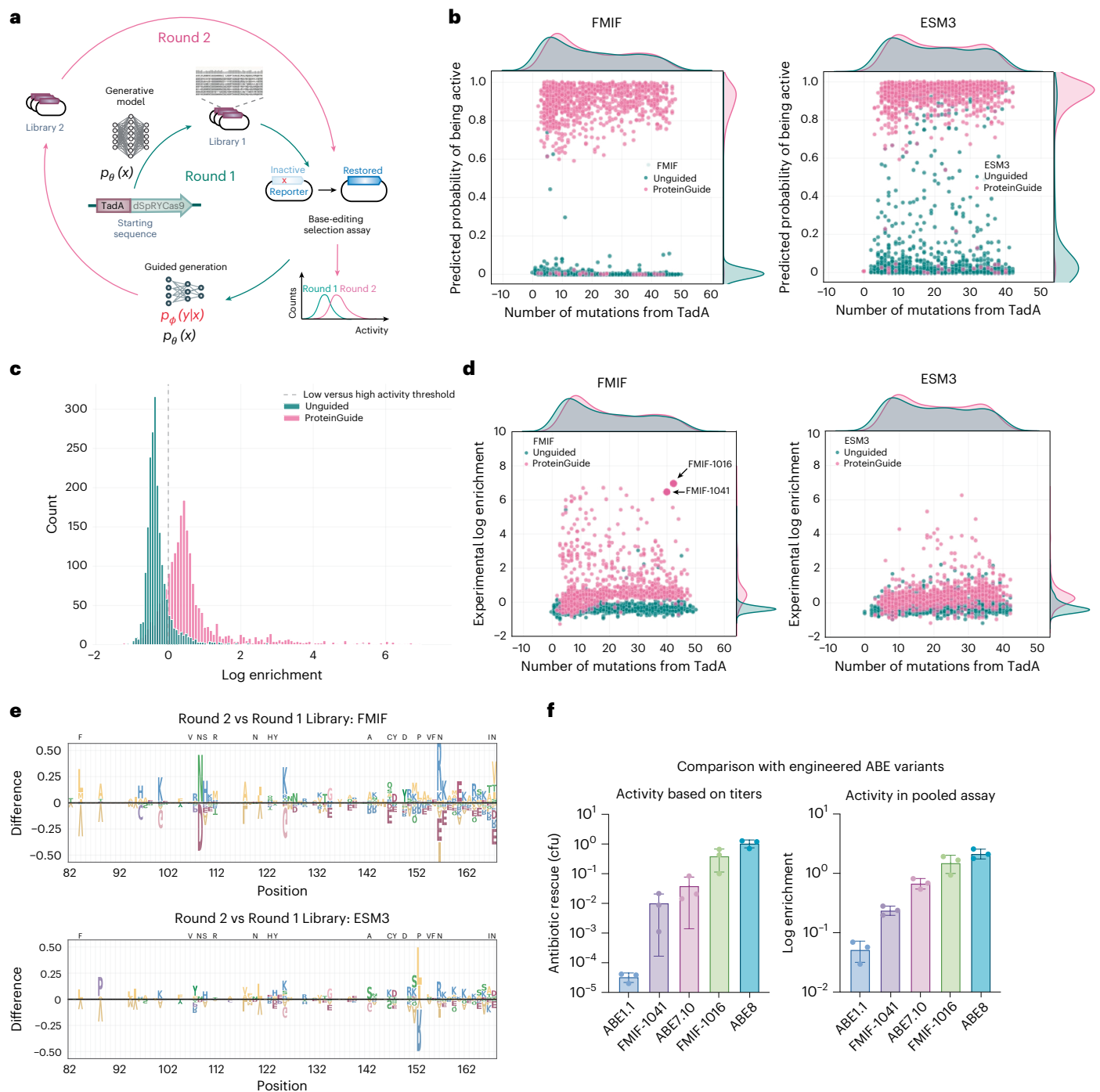


Fig. 5 | Engineering an ABE with ProteinGuide. **a**, Schematic representation of the ProteinGuide ABE experiment. **b**, Scatter plots of the number of mutations from TadA (x-axis) versus the predicted probability of being active (y-axis) for every variant in library 1 (teal) and library 2 (pink) of FMIF (left) and ESM3 (right). **c**, A histogram of the experimentally measured log enrichments for both library 1 (teal) and library 2 (pink). ProteinGuide was used to generate the round 2 library such that variants would be enriched for having high activity (predicted activity in log enrichment units higher than the gray dashed line). **d**, Similar to **b**, but with experimentally measured log enrichments shown on the vertical axis. The two variants selected for further validation are annotated 'FMIF-1016' and 'FMIF-1041'. **e**, Sequence logo plots of the differences in the position-wise amino acid frequencies between library 1 and library 2 for FMIF (top) and ESM3 (bottom). Letters above the horizontal line at $y = 0$ represent

positions or amino acids that are enriched in library 2 relative to library 1 and those below 0 represent positions or amino acids that are depleted in library 2 relative to library 1. Black letters above the plots show substitutions obtained in the experimental evolution campaigns leading to ABE8. **f**, Comparison of the activity of two of the top variants generated by ProteinGuide with experimentally obtained variants with known activities (ABE1.1, ABE7.10 and ABE8). Left, activity estimated on the basis of the titers of individual transformations in the selection strain; shown are colony-forming units (cfu) on the selective normalized to the nonselective condition. Right, activity calculated on the basis of log enrichment in a pooled selection assay measured by sequencing. Bars show the mean values and error bars represent ± 1 s.d. over $n = 3$ independent transformations for each experiment. Individual points show the three independent transformations.

both assays, ProteinGuide's variant with the highest log enrichment value, FMIF-1016, had activity between ABE7.10 and ABE8 (Fig. 5f). Notably, ABE7.10 was found after seven rounds of directed evolution, which included unbiased library diversification, SSM and DNA shuffling⁵⁹. Consequently, one round of ProteinGuide generated a base editor with activity higher than seven rounds of directed evolution. We speculate that further rounds could lead to even higher activity.

Discussion

We introduced ProteinGuide, a broadly applicable framework that enables conditioning of pretrained protein generative models on any property of interest that can be encoded in a predictive model. ProteinGuide does not require retraining the pretrained model or newly training any generative model.

Some of the technical underpinnings for ProteinGuide rely on our earlier development of guidance for continuous-time diffusion and flow-matching models on discrete state spaces, referred to herein as Discrete Guidance¹⁷. Crucially, Discrete Guidance did not address guidance for models other than DDMS^{22,24,27} and DFMS^{25,26}, thereby limiting its utility to the field of protein engineering and beyond. For example, our earlier work did not provide a way to guide ESM3 (ref. 47), ProteinMPNN¹ and many other models^{11,16,51}. With ProteinGuide, we extend guidance to also include MLMs with masking rates covering the range 0–100%, sometimes known as generative MLMs^{64–67} and AO-ARMS^{28,68,69}.

In so doing, ProteinGuide unlocks guidance for a much larger set of model classes and specific existing models relevant to the protein engineering community. For example, ProteinGuide can be used to guide an inverse-folding model to produce proteins with greater solubility, expression or activity than the model naturally generates, to produce proteins more likely to bind a particular target and so forth. ProteinGuide can be used with both existing predictive models for generic properties, such as solubility, expression and activity, and custom models trained on newly acquired experimental measurements. It follows that ProteinGuide should be a key enabler of training-efficient iterative DBTL cycles, allowing users to nimbly incorporate experimental feedback into subsequent artificial-intelligence-based design rounds on the fly.

Through in silico experiments in three distinct design settings—interpolative, extrapolative and multiproperty Pareto extrapolative—we demonstrated the utility of ProteinGuide in a variety of application settings of interest to protein engineers, across a variety of pretrained generative models and spanning numerous protein properties. In our in vivo experiment to engineer an ABE with improved activity, we demonstrated that a single round of ProteinGuide yielded a variant with activity higher than that produced by seven rounds of directed evolution.

Guidance and, hence, ProteinGuide differ from the common practices of post hoc filtering and fine-tuning in a few important ways. First, ProteinGuide uses a separate predictive model to guide sampling from a pretrained generative model, requiring no training or retraining of the generative model itself. Second, it uses the principle of the Bayes rule to statistically, coherently blend information from the pretrained model with the predictive model, thereby enabling generation of variants that the pretrained model might otherwise have been unlikely to produce. Moreover, coherent blending by way of the Bayes rule mitigates the issue of forgetting that often results from fine-tuning^{14,15}.

Beyond straightforward application of our on-the-fly guidance approach, one might consider stacking our ProteinGuide on top of other Bayesian-inspired frameworks that involve retraining a pretrained model toward a reward-weighted posterior distribution^{37–40}. This could be particularly useful in cases where the targeted sequences are substantially out of distribution for the pretrained generative model—the setting where pure guidance would be challenging because the prior used in the Bayes rule would contain too little mass for the

target region. In fact, our experiments on guiding a fine-tuned model point in this methodological direction. Separately, it might prove promising to combine ProteinGuide with MCMC-based methods, allowing multiple ‘particles’ to be kept at each stage of sample generation for one sequence; these methods have been shown to improve the sample quality, albeit at the expense of added sampling computation time^{44–46}. Another possible direction could involve applying ProteinGuide in conjunction with guidance methods for continuous spaces to guide multimodal generative models of both structure and sequence^{25,70}.

Online content

Any methods, additional references, Nature Portfolio reporting summaries, source data, extended data, supplementary information, acknowledgements, peer review information; details of author contributions and competing interests; and statements of data and code availability are available at <https://doi.org/10.1038/s41587-026-03207-z>.

References

- Dauparas, J. et al. Robust deep learning-based protein sequence design using ProteinMPNN. *Science* **378**, 49–56 (2022).
- Watson, J. et al. De novo design of protein structure and function with RFdiffusion. *Nature* **620**, 1089–1100 (2023).
- Ingraham, J. et al. Illuminating protein space with a programmable generative model. *Nature* **623**, 1070–1078 (2023).
- Hsien-Wei Yeh, A. et al. De novo design of luciferases using deep learning. *Nature* **614**, 774–780 (2023).
- Bennett, N. et al. Atomically accurate de novo design of antibodies with RFdiffusion. *Nature* **649**, 183–193 (2026).
- Lisanza, S. et al. Multistate and functional protein design using RoseTTAFold sequence space diffusion. *Nat. Biotechnol.* **43**, 1288–1298 (2025).
- Hong, L. & Kortemme, T. An integrative approach to protein sequence design through multiobjective optimization. *PLoS Comput. Biol.* **20**, e1011953 (2024).
- Ziegler, D. et al. Fine-tuning language models from human preferences. Preprint at <https://doi.org/10.48550/arXiv.1909.08593> (2019).
- Ouyang, L. et al. Training language models to follow instructions with human feedback. In *Proc. 35th Conference on Advances in Neural Information Processing Systems* (eds Oh, A. et al.) (ACM, 2022).
- Ruffolo, J. A. et al. Design of highly functional genome editors by modeling CRISPR–Cas sequences. *Nature* **645**, 518–525 (2025).
- Nijkamp, E., Ruffolo, J., Weinstein, E., Naik, N. & Madani, A. ProGen2: exploring the boundaries of protein language models. *Cell Syst.* **14**, 968–978 (2023).
- Ivančić, D. et al. Discovery and protein language model-guided design of hyperactive transposases. *Nat. Biotechnol.* <https://doi.org/10.1038/s41587-025-02816-4> (2025).
- Munsamy, G. et al. Conditional language models enable the efficient design of proficient enzymes. Preprint at *bioRxiv* <https://doi.org/10.1101/2024.05.03.592223> (2024).
- Kotha, S., Springer, J. & Raghunathan, A. Understanding catastrophic forgetting in language models via implicit inference. In *Proc. 12th International Conference on Learning Representations* (ed. Kim, B.) (ICLR, 2024).
- Luo, Y. et al. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *IEEE Trans. Audio Speech Lang. Process.* **33**, 3776–3786 (2025).
- ESM Team. ESM Cambrian: Revealing the mysteries of proteins with unsupervised learning. *EvolutionaryScale* <https://evolutionaryscale.ai/blog/esm-cambrian> (2024).
- Nisonoff, H., Xiong, J., Allenspach, S. & Listgarten, J. Unlocking guidance for discrete state-space diffusion and flow models. In *Proc. 13th International Conference on Learning Representations* (ed. Yue, Y.) (ICLR, 2025).

18. Dickstein, J., Weiss, E., Maheswaranathan, N. & Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proc. 37th International Conference on Machine Learning* (eds Bach, F. & Blei, D.) (PMLR, 2015).
19. Song, Y. et al. Score-based generative modeling through stochastic differential equations. In *Proc. 9th International Conference on Learning Representations* (ed. Mohamed, S.) (ICLR, 2021).
20. Dhariwal, P. & Nichol, A. Diffusion models beat GANs on image synthesis. In *Proc. 34th Conference on Advances in Neural Information Processing Systems* (eds Ranzato, M. et al.) (ACM, 2021).
21. Ho, J. & Salimans, T. Classifier-free diffusion guidance. In *Proc. Neural Information Processing Systems Workshop on Deep Generative Models and Downstream Applications* (eds Zhang, C. et al.) (NeurIPS, 2021).
22. Campbell, A. et al. A continuous time framework for discrete denoising models. In *Proc. 35th Conference on Advances in Neural Information Processing Systems* (eds Oh, A. et al.) (ACM, 2022).
23. Sun, H., Yu, L., Dai, B., Schuurmans, D. & Dai, H. Score-based continuous-time discrete diffusion models. In *Proc. 11th International Conference on Learning Representations* (ed. Liu, Y.) (ICLR, 2023).
24. Lou, A., Meng, C. & Ermon, S. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Proc. 41st International Conference on Machine Learning* (eds Salakhutdinov, R. et al.) (PMLR, 2024).
25. Campbell, A., Yim, J., Barzilay, R., Rainforth, T. & Jaakkola, T. Generative flows on discrete state-spaces: enabling multimodal flows with applications to protein co-design. In *Proc. 12th International Conference on Machine Learning* (ed. Kim, B.) (ICLR, 2024).
26. Gat, I. et al. Discrete flow matching. In *Proc. 37th Conference on Advances in Neural Information Processing Systems* (eds Zhang, C. et al.) (ACM, 2024).
27. Austin, J., Johnson, D., Ho, J., Tarlow, D. & Berg, R. Structured denoising diffusion models in discrete state-spaces. In *Proc. 34th Conference on Advances in Neural Information Processing Systems* (eds Ranzato, M. et al.) (ACM, 2021).
28. Hoogeboom, E. et al. Autoregressive diffusion models. In *Proc. 9th International Conference on Learning Representations* (ed. Mohamed, S.) (ICLR, 2021).
29. Shi, J., Han, K., Wang, Z., Doucet, A. & Titsias, M. Simplified and generalized masked diffusion for discrete data. In *Proc. 37th Conference on Advances in Neural Information Processing Systems* (eds Zhang, C. et al.) (ACM, 2024).
30. Sahoo, S. et al. Simple and effective masked diffusion language models. In *Proc. 37th Conference on Advances in Neural Information Processing Systems* (eds Zhang, C. et al.) (ACM, 2024).
31. Ou, J. et al. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. In *Proc. 13th International Conference on Learning Representations* (ed. Yue, Y.) (ICLR, 2025).
32. Gordon, C., Lu, A. & Abbeel, P. Protein language model fitness is a matter of preference. In *Proc. 13th International Conference on Learning Representations* (ed. Yue, Y.) (ICLR, 2025).
33. Chen, L. et al. Target sequence-conditioned design of peptide binders using masked language modeling. *Nat. Biotechnol.* **44**, 1002–1010 (2026).
34. Blalock, N. et al. Functional alignment of protein language models via reinforcement learning. Preprint at *bioRxiv* <https://doi.org/10.1101/2025.05.02.651993> (2025).
35. Stocco, F. et al. Guiding generative protein language models with reinforcement learning. Preprint at <https://doi.org/10.48550/arXiv.2412.12979> (2024).
36. Rafailov, R. et al. Direct preference optimization: your language model is secretly a reward model. In *Proc. 36th Conference on Advances in Neural Information Processing Systems* (eds Oh, A. et al.) (ACM, 2023).
37. Widatalla, T., Rafailov, R. & Hie, B. Aligning protein generative models with experimental fitness via direct preference optimization. Preprint at *bioRxiv* <https://doi.org/10.1101/2024.05.20.595026> (2024).
38. Chennakesavalu, S., Hu, F., Ibarraran, S. & Rotskoff, G. Aligning chemical and protein language models with continuous feedback using energy rank alignment. In *Proc. ICLR Workshop on Generative and Experimental Perspectives for Biomolecular Design* (eds Liu, C. et al.) (ICLR, 2025).
39. Wang, C. et al. Fine-tuning discrete diffusion models via reward optimization with applications to dna, and protein design. In *Proc. 13th International Conference on Learning Representations* (ed. Yue, Y.) (ICLR, 2025).
40. Brooks, J. et al. Steering masked discrete diffusion models via discrete denoising posterior prediction. In *Proc. 13th International Conference on Learning Representations* (ed. Yue, Y.) (ICLR, 2025).
41. Wallace, B. et al. Diffusion model alignment using direct preference optimization. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* (ed. Ceballos, C.) (IEEE, 2024).
42. Yang, J. et al. Steering generative models with experimental data for protein fitness optimization. In *Proc. 38th Conference on Advances in Neural Information Processing Systems* (eds Belgrave, D. et al.) (ACM, 2025).
43. Emami, P., Perreault, A., Law, J., Biagioni, D. & John, P. Plug & play directed evolution of proteins with gradient-based discrete MCMC. In *Proc. Neural Information Processing Systems Machine Learning in Structural Biology Workshop* (eds Rao, R. et al.) (NeurIPS, 2022).
44. Wu, L., Trippe, B., Naesseth, C., Blei, D. & Cunningham, J. Practical and asymptotically exact conditional sampling in diffusion models. In *Proc. 36th Conference on Advances in Neural Information Processing Systems* (eds Oh, A. et al.) (ACM, 2023).
45. Li, X. et al. Derivative-free guidance in continuous and discrete diffusion models with soft value-based decoding. In *Proc. 38th Conference on Advances in Neural Information Processing Systems* (eds Belgrave, D. et al.) (ACM, 2025).
46. Singhal, R. et al. A general framework for inference-time scaling and steering of diffusion models. In *Proc. 42nd International Conference on Machine Learning* (eds Singh, A. et al.) (PMLR, 2025).
47. Hayes, T. et al. Simulating 500 million years of evolution with a language model. *Science* **387**, 850–858 (2025).
48. Hsu, C. et al. Learning inverse folding from millions of predicted structures. In *Proc. 39th International Conference on Machine Learning* (eds Chaudhuri, K. et al.) (PMLR, 2022).
49. Alamdari, S. et al. Protein generation with evolutionary diffusion: sequence is all you need. Preprint at *bioRxiv* <https://doi.org/10.1101/2023.09.11.556673> (2023).
50. Wang, X. et al. Diffusion language models are versatile protein learners. In *Proc. 41st International Conference on Machine Learning* (eds Salakhutdinov, R. et al.) (PMLR, 2024).
51. Madani, A. et al. Large language models generate functional protein sequences across diverse families. *Nat. Biotechnol.* **41**, 1099–1106 (2023).
52. Hu, E. J. et al. LoRA: low-rank adaptation of large language models. In *Proc. 10th International Conference on Learning Representations* (eds Hofmann, K. & Rush, A.) (ICLR, 2022).
53. Tsuboyama, K. et al. Mega-scale experimental analysis of protein folding stability in biology and design. *Nature* **620**, 434–444 (2023).
54. Chen, Y. et al. Deep mutational scanning of an oxygen-independent fluorescent protein CreiLOV for comprehensive profiling of mutational and epistatic effects. *ACS Synth. Biol.* **12**, 1461–1473 (2023).

55. Starita, L. et al. Activity-enhancing mutations in an E3 ubiquitin ligase identified by high-throughput mutagenesis. *Proc. Natl Acad. Sci. USA* **110**, E1263–E1272 (2013).
56. Melamed, D., Young, D., Gamble, C., Miller, C. & Fields, S. Deep mutational scanning of an RRM domain of the *Saccharomyces cerevisiae* poly(A)-binding protein. *RNA* **19**, 1537–1551 (2013).
57. Wang, B. et al. Active learning-guided optimization of cell-free biosensors for lead testing in drinking water. *Nat. Commun.* **17**, 261 (2026).
58. Rees, H. & Liu, D. Base editing: precision chemistry on the genome and transcriptome of living cells. *Nat. Rev. Genet.* **19**, 770–788 (2018).
59. Gaudelli, N. et al. Programmable base editing of A•T to G•C in genomic DNA without DNA cleavage. *Nature* **551**, 464–471 (2017).
60. Gaudelli, N. et al. Directed evolution of adenine base editors with increased activity and therapeutic application. *Nat. Biotechnol.* **38**, 892–900 (2020).
61. Richter, M. et al. Phage-assisted evolution of an adenine base editor with improved Cas domain compatibility and activity. *Nat. Biotechnol.* **38**, 883–891 (2020).
62. Xiao, Y., Wu, Y. & Tang, W. An adenine base editor variant expands context compatibility. *Nat. Biotechnol.* **42**, 1442–1453 (2024).
63. Ranzau, B., Rallapalli, K., Evanoff, M., Paesani, F. & Komor, A. The wild-type tRNA adenosine deaminase enzyme TadA is capable of sequence-specific DNA base editing. *ChemBioChem* **24**, e202200788 (2023).
64. Ghazvininejad, M., Levy, O., Liu, Y. & Zettlemoyer, L. Mask-Predict: parallel decoding of conditional masked language models. In *Proc. 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing* (eds Inui, K. et al.) (ACL, 2019).
65. Wang, A. & Cho, K. BERT has a mouth, and it must speak: BERT as a Markov random field language model. In *Proc. Workshop on Methods for Optimizing and Evaluating Neural Language Generation* (eds Bosselut, A. et al.) (ACL, 2019).
66. Chang, H., Zhang, H., Jiang, L., Liu, C. & Freeman, W. MaskGIT: masked generative image transformer. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* (ed. O’Conner, L.) (IEEE, 2022).
67. Li, Y. et al. Promises and pitfalls of generative masked language modeling: theoretical framework and practical guidelines. In *Proc. 41st International Conference on Machine Learning* (eds Salakhutdinov, R. et al.) (PMLR, 2024).
68. Uria, B., Murray, I. & Larochelle, H. A deep and tractable density estimator. In *Proc. 31st International Conference on Machine Learning* (eds Xing, E. P. & Jebara, T.) (PMLR, 2014).
69. Yang, Z. et al. Xlnet: generalized autoregressive pretraining for language understanding. In *Proc. 32nd Conference on Advances in Neural Information Processing Systems* (eds Wallach, H.) (ACM, 2019).
70. Jing, B. et al. Generating functional and multistate proteins with a multimodal diffusion transformer. Preprint at *bioRxiv* <https://doi.org/10.1101/2025.09.03.672144> (2025).
71. Lin, Z. et al. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science* **379**, 1123–1130 (2023).

Publisher’s note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

© The Author(s), under exclusive licence to Springer Nature America, Inc. 2026

Methods

In this section, we first provide an overview of the technical underpinning of ProteinGuide, followed by more detailed technical background. We proceed to show the equivalence of training objectives and sampling algorithms for a broad class of models, including masking-based diffusion and flow-matching models (MDMs and MFMs), generative MLMs and AO-ARMs. Subsequently, full experimental details for all experiments are provided, including an overview for the *in silico* experiments, followed by dedicated descriptions for each experiment.

Overview of ProteinGuide

Conceptually, the problem of guidance involves taking a pretrained generative model, $p(x)$, and a classifier-derived or regression-derived likelihood, $p(y|x)$, to obtain $p(x|y) = p(y|x)p(x)/p(y)$, without retraining the underlying generative model. At its base, designing guidance strategies amounts to exploiting local structure in the generative process so that this reweighting can be implemented tractably and efficiently during generation while producing samples from the correct conditional distribution at the end of the process. As an example, diffusion models on continuous state spaces generate data by learning to reverse a ‘noising’ process, most commonly instantiated as gradually corrupting the data toward pure Gaussian noise^{18,19,72}. Intuitively, guidance in diffusion models is achieved by leveraging a key property of the underlying generative process—namely, that the model learns to map from a simple distribution (for example, a standard normal distribution) to a complex data distribution through a series of infinitesimal steps. Each step ‘points’ toward directions with higher likelihood under the data distribution and can be ‘nudged’, at generation time, toward directions that better satisfy the user-specified likelihood, such that the process ends with samples from the desired conditionals. These infinitesimal steps are formalized as gradients of noise-dependent distributions, known as score functions, and the Bayes rule can be simplified considerably when considering only these gradients, enabling guidance for these models^{18–20}.

For discrete-space generative models, however, these gradients are not defined and such an approach cannot be readily adopted. Nevertheless, discrete-space diffusion and flow-based models also admit a natural local structure, but in a different form. These models learn to reverse a user-defined discrete noising process, such as introducing random mutations or accumulating mask tokens^{27,28}. Such a process can be formalized with continuous-time Markov chains (CTMCs), which can be described as a generative process wherein a state remains unchanged over some amount of holding time and then changes (jumps) to a new state. In practice, these two processes of holding and jumping are all jointly encoded in an object known as the (time-dependent) rate matrix, which describes the time derivative of the probability of transitioning from any state to any other state at a given time. These rate matrices have a similar role to the score functions in continuous-space diffusion models and sampling corresponds to integrating the CTMCs, for example, through straightforward Euler integration²⁵ or with more involved methods^{22,73,74}. To make learning tractable, these models typically assume that the noising processes in each dimension are independent of one another (but not necessarily identical), with the important consequence that, at any given time in the continuous-time noising process, the probability that two or more dimensions of the current state jump (transition) at exactly the same time is zero^{22,25}. Consequently, at any given time, the number of entries in the rate matrix that are nonzero is linear, rather than exponential, in the input space, making such models tractable.

It turns out that the same assumption that makes training these models tractable also makes guidance tractable for them. By reformulating the Bayes rules in terms of the rates, we find that the conditional rates at each sampling step can be obtained by modulating the unguided rates with a likelihood ratio describing how much the guidance signal increases (or decreases) in an infinitesimal time step.

Because, at any given time step, only transitions with a single mutation have nonzero rates, one can tractably compute the normalizing constant for any given time step¹⁷. As this procedure affects only sampling and not training, one can sample from any desired conditional distribution for which a user-specified likelihood is available, without having to retrain the underlying pretrained generative model. We refer to this work as Discrete Guidance and the reader to Nisonoff et al.¹⁷ for more details.

While the framework just presented applies generally to discrete-space diffusion and flow-based models under a broad set of noising processes, one popular instantiation worth considering is a noising process based on masking, where tokens in a sequence are progressively replaced with mask tokens during the noising process and the model learns to recover the original tokens during the denoising process. These models share deep connections to two other popular classes of discrete-space generative models, MLMs with masking rates covering the range 0–100% (sometimes known as generative MLMs) and AO-ARMs. Although MDMs, MFMs, MLMs and AO-ARMs seem to be quite different conceptually, it turns out that the loss functions used for training them are equivalent. While connections among these training objectives have been previously noted^{27–31}, the practical utility of this observation has been unclear. Herein, we show that this connection enables a general framework for guiding all of these model classes. Moreover, we show that sampling from an MDM or MFM can be accomplished more simply, yet exactly, by sampling from a corresponding, trivial-to-construct AO-ARM, which leads to a more efficient, exact guidance algorithm for MDMs, MFMs, MLMs and AO-ARMs. Altogether, our unifying view of training and sampling of all four of these model classes under a single framework enables ProteinGuide to guide a broad class of discrete-space generative models, making it readily applicable to many popular protein sequence generative models commonly used by practitioners.

Technical background

Below, we give a more mathematical treatment of the overview above, providing the technical background important for understanding the results shown.

DDMs and DFMs. While diffusion models were initially developed for continuous state spaces^{18,19,72}, recent work has extended these frameworks to discrete domains such as text, molecular graphs and protein sequences. These extensions enable generative modeling of inherently discrete data without requiring continuous relaxations. There are a number of works defining discrete-time diffusion processes in discrete spaces^{27,75}. They were later extended to continuous-time processes^{22–24} leveraging the formulation of CTMCs, which describe stochastic processes where a system transitions between discrete states over continuous time. The dynamics of a CTMC are fully determined by an initial distribution $p(x_0)$ and transition rates $R_t(x, \bar{x})$ that specify the instantaneous probability of transitioning from state x to state $\bar{x}$ at time t :

$$p(X_{t+dt} = \bar{x} | X_t = x) = \delta_{x,\bar{x}} + R_t(x, \bar{x})dt + O(dt^{1+\epsilon}). \quad (1)$$

As an example, Campbell et al.²² defined a forward noising process by specifying a time-dependent ‘forward’ rate matrix that gradually transforms the (unknown) data distribution into a reference distribution, such as a uniform distribution over amino acid sequences. A denoising model is then trained to reverse this process, learning rates that can transform the corrupted samples back to samples from the data distribution.

There are also a number of studies that extend flow matching, which were originally developed for continuous state-spaces^{76–78}, to discrete state-spaces^{25,26}, also leveraging CTMCs, but differ from diffusion models in how the CTMCs are defined, which also induce different training objectives. These models specify a conditional flow that

defines how probability mass should move from a noise distribution to the data distribution. The model learns to generate samples by following the flow trajectories. Campbell et al.²⁵ showed that the rates used in generative sampling can be obtained by training a neural network with parameters θ , $p_\theta(x_1|x_t, t)$ to approximate the true denoising distribution using the standard cross-entropy loss,

$$\mathcal{L}_{\text{FM}} = \mathbb{E}_{x_1 \sim p_{\text{data}}(x_1), t \sim p(t), x_t \sim p_{t|1}(x_1|x_1)} [w_t(x_t) \log p_\theta(x_1|x_t, t)], \quad (2)$$

where x_1 is the clean data, $p(t)$, referred to as the noise schedule, is a distribution with full support on the interval $[0, 1]$ and x_t is the noised data sample from the forward noising process $p_{t|1}(x_t|x_1)$. For all models, we follow the convention of flow-based models, where $t = 1$ is the target (training) distribution and $t = 0$ is the noise distribution. As shown in Campbell et al.²⁵, the weighted cross-entropy loss can be understood as a simplification of the evidence lower bound (ELBO) used to train continuous-time DDMS²². $w_t(x_t)$ denotes a time-dependent weighting function and a common choice is to set $w_t(x_t) = 1$ (ref. 25). Different choices of this weighting correspond to different but closely related diffusion or flow-matching training objectives, which, for continuous-space diffusion models, can be shown to be equivalent to the ELBO combined with simple data augmentation⁷⁹.

Given a trained denoising model $p_\theta(x_1|x_t, t)$, the rate matrix used in sampling is defined as

$$R_t^\theta(x_t, \tilde{x}) = \mathbb{E}_{x_1 \sim p_\theta(x_1|x_t, t)} [\vec{R}_t(x_t, \tilde{x}|x_1)], \quad (3)$$

where $\vec{R}_t(x_t, \tilde{x}|x_1)$ is the data-conditional flow rate induced by the probability path $p_{t|1}(x_t|x_1)$ and $\tilde{x}$ denotes a possible next state reachable from x_t .

In practice, for computational tractability, an independent noising process is used for each dimension, $i \in \{1, 2, \dots, D\}$, of x_t . Campbell et al.^{22,25} showed that, in this setting, the denoising model, $p_\theta(x_1|x_t, t)$, need only learn a factorized distribution of the clean data and the loss becomes

$$\mathcal{L}_{\text{FM}} = \mathbb{E}_{x_1 \sim p_{\text{data}}(x_1), t \sim p(t), x_t \sim p_{t|1}(x_t|x_1)} \left[w_t(x_t) \sum_{i=1}^D \log p_\theta(x_t^i|x_t, t) \right]. \quad (4)$$

Similarly, the rate matrix can be expressed per dimension as

$$R_t^{\theta,i}(x_t, \tilde{x}^i) = \mathbb{E}_{x_1^i \sim p_\theta(x_1^i|x_t, t)} [\vec{R}_t^i(x_t^i, \tilde{x}^i|x_1^i)], \quad (5)$$

where $\vec{R}_t^i(x_t^i, \tilde{x}^i|x_1^i)$ is the data-conditional flow rate for dimension i induced by the one-dimensional probability path $p_{t|1}(x_t^i|x_1^i)$.

There are many methods for sampling from discrete state-space diffusion and flow models by integrating the learned rate matrices with numerical integration schemes^{22,25,26,80}. One simple approach is Euler integration with a discrete time step Δt , where transition probabilities are computed as $p(x_{t+\Delta t} = \tilde{x}|x_t = x) = \delta_{x, \tilde{x}} + R_t(x, \tilde{x})\Delta t$ and new states are sampled categorically at each step. Alternative methods include Gillespie's algorithm^{73,74} for exact simulation and τ -leaping²² for approximate sampling that allows multiple dimensions to transition simultaneously. The flow-matching formulation also introduces a stochasticity hyperparameter η that can be adjusted at inference time to control the randomness of the sampling path, which Campbell et al.²⁵ showed could improve sample quality.

A popular instantiation of both discrete state-space diffusion and flow models uses masking as the per-dimension noising process, where tokens in a sequence are progressively replaced with mask tokens during the forward process and the model learns to recover the original tokens during the reverse process. For example, the masking conditional flow can be defined as $p^{\text{mask}}(x_t^i|x_1^i) = \text{Cat}(t\delta[x_t^i, x_t^i] + (1-t)\delta[?, x_t^i])$, where $\delta[i, j]$ is the Kronecker delta, which is 1 when $i = j$ and 0 otherwise,

and '?' denotes a mask token. Campbell et al.²⁵ showed that, under the masking noise process, the unconditional rate matrix used in sampling defined in Eq. (5) simplifies to

$$R_t^{\theta,i}(x_t, \tilde{x}^i) = \frac{p_\theta(x_1^i = \tilde{x}^i|x_t, t)}{1-t} \delta[x_t^i, ?], \quad \tilde{x}^i \neq x_t^i. \quad (6)$$

For the masking noise process, Campbell et al.²⁵ further showed that the training objective reduces to a weighted cross-entropy over only the masked positions:

$$\mathbb{E}_{x_1 \sim p_{\text{data}}(x_1), t \sim \text{Unif}([0,1]), x_t \sim p_{t|1}(x_t|x_1)} \left[\frac{1}{1-t} \sum_{i \in M_t} \log p_\theta(x_1^i|x_t, t) \right] \quad (7)$$

where $M_t = \{i : x_t^i = ?\}$ denotes the set of masked positions, $1-t$ is the probability that a position is masked under the masking noise process and the factor $(1-t)^{-1}$ is the corresponding inverse mask-probability weighting. In practice, it is common to train with a simpler, uniformly weighted objective, averaging the cross-entropy over the masked positions^{25,27}:

$$\mathcal{L}_{\text{MFM}} = \mathbb{E}_{x_1 \sim p_{\text{data}}(x_1), t \sim \text{Unif}([0,1]), x_t \sim p_{t|1}(x_t|x_1) | |M_t| > 0} \left[\frac{1}{|M_t|} \sum_{i \in M_t} \log p_\theta(x_1^i|x_t, t) \right]. \quad (8)$$

This formulation has connections to MLMs and AO-ARMs, which we describe below in detail.

MLMs. MLMs loosely refer to a broad class of models that involve training a model to recover sequences of discrete tokens given the input where some of the tokens have been replaced by a special token known as the 'mask' token. Herein, we focus on a class of MLMs that we refer to as generative MLMs^{64,66}. The key distinction between generative MLMs and their nongenerative counterparts⁸¹ is that, during training, generative MLMs use a variable masking probability sampled from a distribution with support on the interval $[0, 1]$. In our notations, t denotes the probability that a token remains unmasked; thus, so $1-t$ is the masking probability. This requirement is met by MLMs such as ESM3 (ref. 47); however, importantly, is not met by bidirectional encoder representations from transformers model (BERT)-style MLMs such as ESM2 (ref. 71), as these models use a fixed percentage of tokens that are masked at each step.

Letting x_t refer to the masked sequence with masking proportion denoted by t and $M_t = \{i : x_t^i = ?\}$ denote the set of masked positions, we write the generative MLM objective function as

$$\mathcal{L}_{\text{MLM}} = \mathbb{E}_{x_1 \sim p_{\text{data}}(x_1), t \sim p(t), x_t \sim p_{t|1}(x_t|x_1) | |M_t| > 0} \left[\frac{1}{|M_t|} \sum_{i \in M_t} \log p_\theta(x_1^i|x_t, t) \right]. \quad (9)$$

There are various approaches that are used to sample from a generative MLM. Some common approaches include directly decoding all the tokens from the fully masked sequence or decoding the sequence iteratively⁴⁷. For iteratively decoding, a commonly used heuristic is to choose the decoding order that prioritizes the token to be unmasked according to some confidence⁶⁶.

AO-ARMs. AO-ARMs differ from standard autoregressive models by training on arbitrary permutations of the sequence ordering^{28,68}. Consider a sequence $x = (x^1, x^2, \dots, x^D)$ from our data distribution. In an AO-ARM, we sample a permutation σ of indices $\{1, 2, \dots, D\}$ and factorize the joint distribution as

$$p_\theta(x) = \prod_{i=1}^D p_\theta(x^{\sigma(i)}|x_1^{\sigma(<i)}) \quad (10)$$

where $x^{o(i)}$ denotes $(x^{o(1)}, x^{o(2)}, \dots, x^{o(i-1)})$. The AO-ARM objective is to maximize the expected log likelihood over all possible permutations:

$$\mathcal{L}_{\text{AO-ARM}} = \mathbb{E}_{x \sim p_{\text{data}}(x), \sigma \sim \text{Unif}(\text{Perm}(D))} \left[\sum_{i=1}^D \log p_{\theta}(x^{o(i)} | x^{o(<i)}) \right] \quad (11)$$

Sampling from an AO-ARM^{28,68,69,82} involves iteratively constructing a sample, $x = (x^1, \dots, x^D)$ of length D , starting from a fully masked sequence, by first sampling an ordering of positions uniformly at random and then sampling iteratively from the conditional distribution of the model given this ordering.

Conditional generation with guidance. Guidance is a useful technique that enables conditional generation with diffusion and flow models^{18–20}. Originally developed for continuous state-space models, guidance allows steering the generative process toward samples with desired properties without requiring conditional training. In continuous state-space diffusion models, the generation process involves sampling from a reverse stochastic differential equation that gradually transforms noise into data¹⁹. To condition this process on a desired property y , it has been shown that one can leverage the Bayes rule to obtain the conditional score function¹⁹:

$$\nabla_{x_t} \log p_t(x_t | y) = \nabla_{x_t} \log p_t(x_t) + \nabla_{x_t} \log p_t(y | x_t) \quad (12)$$

This formulation, termed classifier guidance, combines the unconditional score $\nabla_{x_t} \log p_t(x_t)$ with the gradient of a predictor model $\nabla_{x_t} \log p_t(y | x_t)$. The strength of guidance can be controlled by introducing a guidance strength parameter γ ²⁰:

$$\nabla_{x_t} \log p_t^{(\gamma)}(x_t | y) = \nabla_{x_t} \log p_t(x_t) + \gamma \nabla_{x_t} \log p_t(y | x_t) \quad (13)$$

This approach provides several advantages. Firstly, it enables conditioning an unconditional model without retraining. Secondly, the guidance strength γ can be adjusted at inference time. Thirdly, it allows composition of multiple guidance signals. Ho and Salimans²¹ later introduced classifier-free guidance, which achieves similar results without an explicit classifier by combining a conditionally trained model and an unconditional model, which in practice are typically trained jointly. Classifier-free guidance has been shown to improve sample quality²¹ compared to using only the conditionally trained model and recent work on autoguidance showed further improved sample quality by combining a trained model with a more inferior version of itself⁶³.

While guidance, whether classifier-based guidance, classifier-free guidance or autoguidance, has proven extremely valuable for continuous state-space models, extending these concepts to discrete state spaces poses unique challenges, as the score functions are not defined and these formulations cannot be directly applied. One can sidestep the need for a score function by instead returning to the Bayes rule, which, in the context of diffusion on discrete state spaces, dictates that we must effectively compute

$$p(x_{t+dt} | x_t, y) = \frac{p(y | x_{t+dt}, x_t) p(x_{t+dt} | x_t)}{\sum_{x'_{t+dt}} p(y | x'_{t+dt}, x_t) p(x'_{t+dt} | x_t)}, \quad (14)$$

where x_{t+dt} and x_t are separated by a infinitesimal time step dt and y is the desired property we wish to condition on. In a D -dimensional discrete state space where each dimension has cardinality S , there are S^D terms in the normalizing constant of Eq. (14), rendering it generally intractable. Each term arises from the fact that, without constraints, any state can be reached from any other state. Intuitively, tractability can only be achieved if the number of terms is not exponential, thereby implying some constraints on the allowed state transitions. Next, we see how the continuous-time formulation of discrete state-space

diffusion and flow models enables us to tractably compute the normalizing constant by imposing such constraints without losing any model expressibility.

Guidance in discrete state spaces. Here, we provide an overview for Discrete Guidance, a general method for guidance in discrete-space diffusion and flow models, and refer the readers to Nisonoff et al.¹⁷ for detailed derivations and additional results. To enable guidance for discrete state-space models, we turn to continuous-time diffusion models and flow models in discrete state-spaces. As described above, these models are based on CTMCs and, therefore, parameterize transition rates $R_t(x, \bar{x})$ rather than score functions. These models also assume that the noising processes in each dimension are independent of one another (but not necessarily identical), which has the important consequence that the probability that two or more dimensions of the current state jump (transition) at exactly the same time is zero, which makes these models tractable^{22,25}. It turns out that the same assumption also makes guidance tractable for these models¹⁷.

To perform predictor guidance in this setting, we derived a principled approach to adjust these rates on the basis of a predictor model $p(y | x_t)$ that evaluates how likely a state x_t is to have the desired property y ¹⁷. Specifically, we showed that the conditional rates can be obtained as

$$R_t(x, \bar{x} | y) = \frac{p(y | \bar{x}, t)}{p(y | x, t)} R_t(x, \bar{x}) \quad (15)$$

for any state $\bar{x}$ that differs from x in at most one dimension. This formulation directly implements the Bayes rule at the level of transition rates, modulating transitions toward states that have higher likelihood of possessing the desired property.

Similar to continuous state-space guidance, we can control the strength of guidance with a guidance strength parameter γ ,

$$R_t^{(\gamma)}(x, \bar{x} | y) = \left[\frac{p(y | \bar{x}, t)}{p(y | x, t)} \right]^\gamma R_t(x, \bar{x}) \quad (16)$$

or derive predictor-free guidance discrete state spaces,

$$R_t^{(\gamma)}(x, \bar{x} | y) = R_t(x, \bar{x} | y)^\gamma R_t(x, \bar{x})^{1-\gamma} \quad (17)$$

For computational efficiency, especially with large state spaces, we introduced TAG¹⁷, which approximates the likelihood ratio using a first-order Taylor expansion inspired by Grathwohl et al.⁸⁴:

$$\log \frac{p(y | \bar{x}, t)}{p(y | x, t)} \approx (\bar{x} - x)^T \nabla_x \log p_\theta(y | x, t) \quad (18)$$

requiring only one forward and one backward pass of the predictor model instead of $D \times (S - 1) + 1$ forward passes, making estimation of the guide-adjusted rates $O(1)$ rather than $O(D \times S)$ at each sampling step. With multiple particles, TAG can be further improved by acting as a proposal distribution in a sequential Monte Carlo algorithm similar to Wu et al.⁴⁴. While there are other recently proposed approaches for guiding DDMs^{45,46,85}, ProteinGuide has the benefit of being both exact and general, applicable to both diffusion and flow-based models, making it a general and flexible framework for guiding a broad class of discrete state-space generative models (discussion on related work in the Supplementary Text).

Unified connections of training objectives for masking-based generative models

Connections among masked DDMs, MLMs and AO-ARMs were initially pointed out by Austin et al.²⁷ and Hooeboom et al.²⁸. Recent studies^{25,26,29–31} further clarified these connections, including the connections to CTMC-based formulation of diffusion and flow models^{22,24,25}, and showed that the continuous-time ELBO used to train DDMs and

MFMs can essentially be reduced to a weighted sum of cross-entropy losses predicting the masked states from unmasked positions, where the proportion of masked positions ranges from complete masking to no masking. For particular choices of the time-dependent weighting functions, these training objectives can be shown to be equivalent to the training objective of AO-ARMs⁶⁸, (generative) MLMs²⁷ and DFMs^{25,26} under a masking process.

Because ESM3 (ref. 47) and ProteinMPNN¹ were trained under the MLM and AO-ARM objectives, respectively, and because of the sampling equivalency between AO-ARMs and diffusion models, as discussed below, we can view ESM3 and ProteinMPNN as discrete state-space diffusion and flow models and sample from them as such, leveraging the generative rate matrices from Eq. (3) induced by the trained denoising model, $p_\theta(x_t|x_t)$. This formulation not only provides a principled way to sample from these models but also offers additional sampling flexibility, including the usage of different noise schedules during sampling from those used during training and corrector sampling^{25,26}, which have been shown to sometimes improve sampling results compared to prior heuristics⁶⁶ for sampling from generative MLMs²⁶. In addition, this formulation unlocks the capacity to condition the generative process through guidance, which we introduce in a later section.

Here, we provide a self-contained derivation of the equivalence between the loss functions of masked flow matching (MFMs), MLMs and AO-ARMs. We also refer the readers to other related studies^{25–31} for more detailed derivations that emphasize certain aspects of the arguments outlined here.

Equivalence between MFMs and MLMs. For the masking noise process, only the masked positions contribute nontrivial denoising targets. Therefore, we consider the mask-averaged masking flow-matching objective

$$\mathcal{L}_{\text{MFM}} = \mathbb{E}_{x_1 \sim p_{\text{data}}(x_1), t \sim p(t), x_t \sim p_{\text{q1}}(x_t|x_1) | |M_t| > 0} \left[\frac{1}{|M_t|} \sum_{i \in M_t} \log p_\theta(x_t^i|x_t, t) \right].$$

We now show that, when a masking noise process is used, this denoiser's dependency on t can be dropped, making the mask-averaged MFM objective equivalent to the MLM objective in Eq. (9).

Let $x_1 = (x_1^1, x_1^2, \dots, x_1^D)$ be our original sequence from the data distribution. For a masking noise process, we define binary masking variables $m_t^i \in \{0, 1\}$ for each position i at time t , where $m_t^i = 1$ indicates that token i is masked at time t and $m_t^i = 0$ indicates that token i is not masked at time t . Using '?' to denote a masked state, the state x_t is then defined as

$$x_t^i = \begin{cases} x_1^i & \text{if } m_t^i = 0 \\ ? & \text{if } m_t^i = 1. \end{cases} \quad (19)$$

Let us additionally define $M_t = \{i : x_t^i = ?\}$ as the set of masked positions at time t and $U_t = \{i : x_t^i \neq ?\}$ as the set of unmasked positions. For any position $i \in U_t$, we know with certainty that $x_t^i = x_1^i$ by the definition of our forward process.

The key observation is that the time variable t only influences which tokens are masked (the masking pattern); however, once we observe x_t , we have complete information about which positions are masked and which are not. The time variable, therefore, provides no additional information needed to predict the clean data. Using a mathematical argument first proposed by Gat et al.²⁶, we can prove this as follows:

$$p(x_1|x_t, t) = \frac{p(x_t|x_1, t)p(x_1)}{\sum_{\tilde{x}_1} p(x_t|\tilde{x}_1, t)p(\tilde{x}_1)} \quad (20)$$

$$= \frac{\prod_{i \in M_t} (1-t) \prod_{i \in U_t} t \left[\prod_{i \in U_t} \delta(x_t^i, x_1^i) \right] p(x_1)}{\sum_{\tilde{x}_1} \left[\prod_{i \in M_t} (1-t) \prod_{i \in U_t} t \right] \left[\prod_{i \in U_t} \delta(x_t^i, \tilde{x}_1^i) \right] p(\tilde{x}_1)} \quad (21)$$

$$= \frac{\left[\prod_{i \in U_t} \delta(x_t^i, x_1^i) \right] p(x_1)}{\sum_{\tilde{x}_1} \left[\prod_{i \in U_t} \delta(x_t^i, \tilde{x}_1^i) \right] p(\tilde{x}_1)} \quad (22)$$

$$= p(x_1|x_t). \quad (23)$$

where Eq. (21) uses the factorization over the dimensions and analyzes separately the two cases of whether a position is masked or unmasked, while Eq. (22) shows that $p(x_1|x_t, t)$ does not depend on t , giving us Eq. (23). Thus, we can rewrite the masking flow-matching loss function as

$$\mathcal{L}_{\text{MFM}} = \mathbb{E}_{x_1 \sim p_{\text{data}}(x_1), t \sim p(t), x_t \sim p_{\text{q1}}(x_t|x_1) | |M_t| > 0} \left[\frac{1}{|M_t|} \sum_{i \in M_t} \log p_\theta(x_t^i|x_t, t) \right] \quad (24)$$

$$= \mathbb{E}_{x_1 \sim p_{\text{data}}(x_1), t \sim p(t), x_t \sim p_{\text{q1}}(x_t|x_1) | |M_t| > 0} \left[\frac{1}{|M_t|} \sum_{i \in M_t} \log p_\theta(x_t^i|x_t) \right] \quad (25)$$

$$= \mathcal{L}_{\text{MLM}}. \quad (26)$$

Equivalence between MFMs and AO-ARMs. Recall from Eq. (11) that the AO-ARM objective is to maximize the expected log likelihood over all possible permutations:

$$\mathcal{L}_{\text{AO-ARM}} = \mathbb{E}_{x \sim p_{\text{data}}(x), \sigma \sim \text{Unif}(\text{Perm}(D))} \left[\sum_{i=1}^D \log p_\theta(x^{\sigma(i)}|x^{\sigma(<i)}) \right]$$

We show that this objective can be equivalently expressed as a uniformly weighted, mask-averaged masking flow-matching loss:

$$\mathcal{L}_{\text{AO-ARM}} = \mathbb{E}_{x_1 \sim p_{\text{data}}(x_1), \sigma \sim \text{Unif}(\text{Perm}(D))} \left[\sum_{i=1}^D \log p_\theta(x_1^{\sigma(i)}|x_1^{\sigma(<i)}) \right] \quad (27)$$

$$= D \mathbb{E}_{x_1 \sim p_{\text{data}}(x_1), \sigma \sim \text{Unif}(\text{Perm}(D)), t \sim \text{Unif}(\{1, \dots, D\})} \left[\log p_\theta(x_1^{\sigma(i)}|x_1^{\sigma(<i)}) \right] \quad (28)$$

$$= D \mathbb{E}_{x_1 \sim p_{\text{data}}(x_1), t \sim \text{Unif}(\{0, 1\}), x_t \sim p_{\text{q1}}(x_t|x_1) | |M_t| > 0, j \sim \text{Unif}(M_t)} \left[\log p_\theta(x_t^j|x_t) \right] \quad (29)$$

$$= D \mathbb{E}_{x_1 \sim p_{\text{data}}(x_1), t \sim \text{Unif}(\{0, 1\}), x_t \sim p_{\text{q1}}(x_t|x_1) | |M_t| > 0} \left[\frac{1}{|M_t|} \sum_{j \in M_t} \log p_\theta(x_t^j|x_t) \right] \quad (30)$$

which is D times the mask-averaged MFM objective $\mathcal{L}_{\text{MFM}}$ evaluated under the uniform schedule, $p(t) = \text{Unif}(\{0, 1\})$. The first step rewrites the sum over the D decoding steps as D times an expectation over a uniformly sampled decoding step. To go from Eq. (28) to Eq. (29), note that sampling $i \sim \text{Unif}(\{1, \dots, D\})$ makes the number of unrevealed positions $D - i + 1$ uniform over $\{1, \dots, D\}$. Conditioned on this number, the unrevealed set is a uniformly random subset of positions and the next target position $\sigma(i)$ is uniformly distributed over this set. Equivalently, under $t \sim \text{Unif}(\{0, 1\})$ and the masking noise process, the number of masked positions $|M_t|$ is uniform over $\{0, \dots, D\}$; conditioning on $|M_t| > 0$ gives a uniform distribution over $\{1, \dots, D\}$. Given x_t , the target position is uniformly distributed over M_t , which gives the average over masked positions. Thus, the AO-ARM objective is exactly equal, up to the constant D , to the uniformly weighted, mask-averaged masking flow-matching objective.

Simplified and faster sampling for flow-matching and diffusion models with masking noise processes

Above, we showed an equivalence among the loss functions used to train DFMs, DDMs, AO-ARMs and MLMs, specifically when the DFMs and DDMs used a masking noise process, which we referred to as MFMs

and MDMs. Next, we show a relationship among sampling algorithms used for these models.

Intuitive overview. We first provide an intuitive overview of the main technical result. We show that, instead of sampling from a CTMC—the required operation for sampling from a CTMC-based DFM—by way of integration in time using algorithms such as Euler, Gillespie or τ -Leaping (of which only Gillespie is exact), one can use a much simpler algorithm to exactly sample from the CTMC. Specifically, one can use a simpler sampling algorithm that is normally used to sample from an AO-ARM. In particular, we are able to exactly sample from a CTMC underlying a DFM by independently sampling all transition times up front (that is, before any state transitions have been sampled) and only then sampling all the state transitions. This lies in stark contrast to general CTMC sampling, in which sampling time and state transitions must be interleaved. However, it is exactly how one would sample from an AO-ARM, where one first samples a ‘decoding order’ and then autoregressively unmask each position in turn according to this order, conditioning on all previously unmasked positions. Consequently, there is constructive proof below for AO-ARM to sample from so as to sample from the desired CTMC associated with a DFM. As a corollary, we also show that one only needs the ordering of the transition times to sample the state transitions.

To achieve correctness of this simplified sampling algorithm, we need to make use of commonly used modeling choices made for CTMC-based DFMs, such that these CTMCs follow an interpolation schedule as prescribed by the flow-matching setup. These choices enable us to derive a joint probability distribution for the CTMC over time and sequences that factorizes in the necessary way, that is, into a term containing solely transition (holding) times and another containing solely state transitions.

So far, we have only discussed DFMs; however, through other work that connects DFMs as a generalization of DDMs, these diffusion models inherit all the work of sampling from a DFM. We discuss related work for DFMs below. Because we showed earlier that the training objective for MLMs is the same as for AO-ARM and MDMs, we can similarly appeal to our earlier reasoning of interpreting an MLM as either an AO-ARM or a DFM and, correspondingly, also sample from it in the same way as described herein.

In addition to showing simplified and faster sampling for CTMC-based DFMs, we also show how to use this new viewpoint to provide an alternative derivation for the guidance results for CTMC-based diffusion and flow-matching models presented in Nisonoff et al.¹⁷

Related work on sampling algorithms. A number of existing studies have discussed the topics herein but these were for the specific case of diffusion models, namely CTMC-based MDMs. We prove similar results for the case of flow-matching models, namely CTMC-based MFMs. We also synthesize these results from prior work to give coherent, constructive proof for the equivalence of paths sampled by MFMs and AO-ARM.

As mentioned above, it has been previously shown that sampling exactly from the CTMC of a MDM is equivalent to sampling unmasking times for each position up front and then using this to define the decoding order to sample from an AO-ARM^{28–31,80,86,87}. The discussions in Ou et al.³¹ and Peng et al.⁸⁰ are most similar to this section in spirit but, again, analyze the equivalence for MDMs specifically. Furthermore, neither work gives unified proof of the equivalence in sampling that allows complete removal of the notion of time when sampling from an MFM, showing equivalence for the full path and not just marginal distributions. Appendix H of Campbell et al.²⁵ further shows a correspondence between DDMs and DFMs, where DFMs define a large class of rate matrices for which DDMs correspond to a specific choice. We believe that this holds in the general case, such that our results for MFMs should be reducible to the previously cited results for MDMs.

Previous studies by Campbell et al.²⁵ and Gat et al.²⁶ on MFMs mentioned that the conditional distribution parameterized by the neural network is independent of time (analogous MDM results in other studies^{29,30,80}). This is a key insight that we use herein. However, as used by prior work, we do not believe that this insight suffices to show that the AO-ARM sampling method for MDMs samples the desired CTMC, nor does it directly extend the MDM proofs to MFMs. Herein, we provide proof of the exact equivalence, which additionally gives a method to sample the transition times. Integrating these findings, Proposition 1 gives us constructive proof to sample from the desired CTMC using either an MFM or an AO-ARM.

Our proofs hold for the common MFM formulation from Campbell et al.²⁵ and Gat et al.²⁶, which we define again below. Many of the results can readily be extended to other noise distributions, such as uniform substitutions, or to the use of stochasticity and corrector sampling. However, in those cases, a full reduction to AO-ARM sampling is not possible. Instead, the calculations yield an efficient Gillespie sampling algorithm, which we do not cover in detail.

Background and notation. Below, we briefly review the relevant concepts for AO-ARMs, CTMCs and flow-matching models. Although some of these were covered earlier, here, we quickly remind the reader so as to also introduce specifically relevant notation.

Sampling from an AO-ARM^{28,68,69,82} involves iteratively constructing a sample of length D , $X = (X^1, \dots, X^D)$, starting from a fully masked sequence, in two steps. First, a permutation, σ , of positions, $1, 2, \dots, D$ is sampled uniformly at random. This permutation dictates the order of positions to unmask as the sample is constructed over D iterations (one for each position), sometimes called the decoding order. Given the permutation ordering, $\sigma(1, 2, \dots, D)$, the model progressively unmasks each position according to this ordering, one at a time, autoregressively. That is, at each step, the model samples from the conditional distribution, $X^{\sigma(i)} \sim p(X^{\sigma(i)} | X^{\sigma(<i)})$. These distributions are the core object learned when training an AO-ARM, typically by a neural network. We denote the set of progressively unmasked sequences generated during the autoregressive sampling as $\mathbf{X} = (X_1, \dots, X_D)$, where bold indicates a ordered set of random variables and subscript denotes the i th unmasking iteration. Note that the dimension of X is D and there are D unmasking operations to generate it, which can be confusing. We sometimes refer to $\mathbf{X}$ as the ‘path’. We can also represent the path compactly as a tuple of the decoding order and the final sequence, denoted $\mathbf{X} = (\sigma, X)$.

Formally, we write the joint probability of the permutation ordering, σ , and the fully unmasked sequence, X (together sometimes called the ‘decoding path’), as

$$\mathbb{P}_{\text{OA}}(\sigma, X) = U[S_D](\sigma) \prod_{i=1}^D p(X^{\sigma(i)} | X^{\sigma(<i)}), \quad (31)$$

where S_D is the group of permutations over sequences of length D , $U[S_D](\sigma)$ indicates that σ is sampled uniformly at random from S_D , $X^{\sigma(i)}$ is the random variable for the value of position $\sigma(i)$ and $X^{\sigma(<i)}$ is the set of $i-1$ observed positions that are unmasked at the i th decoding step.

CTMCs describe a sequence of Markovian transitions between discrete states as a continuous-time process. A detailed introduction, including path probability calculations like those performed here, was provided by Del Moral and Penev⁸⁸. When a CTMC ‘jumps’ from a state x to state $\bar{x}$ at time τ , we say that the CTMC experienced a state transition from x to $\bar{x}$ at jump time τ . As the transition is instantaneous, we define the new state to be the state at time τ : $X_\tau = \bar{x}$. Accordingly, we say that the previous state x is the left limit of the CTMC at time τ , meaning that it is the value of the CTMC going right up to the jump time. We denote the left limit as $X_{\tau-} = \lim_{t \nearrow \tau} X_t = X_{\tau_{-1}}$.

We often omit the random variables when the symbol for the observed value is the corresponding lowercase letter. For example, $\mathbb{P}_{\text{MFM}}(x_t) = \mathbb{P}_{\text{MFM}}(X_t = x_t)$ refers to the probability of observing the

random variable X_t taking on value x_t . Sometimes, when the variable for the observed value does not correspond to the random variable it is assigned to, we write both explicitly. For example, in writing $\mathbb{P}_{\text{MFM}}(X_t | X_t^- = x_s)$, we refer to the probability that the MFM transitions from x_s , the state it reached at a previous time s , to x_t at time t . Recall that $X_t^- = x_s$ indicates that the state x_s is the left limit X_t^- of the state at t .

Lastly, the rate matrix from x to $\bar{x}$ at time t is denoted $R_t(x, \bar{x})$. The rate matrix is generally time dependent even when conditioned on the state. In flow-matching models, transition rates between states generally accelerate in time to ensure that the process terminates by time $t = 1$. Accordingly, the CDF of the jump time is an integral of time-dependent rates and does not always have a closed-form expression. It is written as

$$\mathbb{P}(T_k < t | T_{k-1}) = 1 - \exp\left(\int_{s=T_{k-1}}^{s=t} R_s(X_s^-, X_s^-) ds\right). \quad (32)$$

As one can see, this integrates the rate of the CTMC not undergoing a state transition at time s .

DFMs were formulated by Campbell et al.²⁵ and Gat et al.²⁶ to define a CTMC whose marginal distribution over states interpolates between a noise distribution p_{noise} at time 0 and a data distribution p_{data} at time 1. Formally, this requires that the rates learned by our DFMs are solutions to the Kolmogorov forward equation (KFE) that satisfy the interpolation $X_t = \kappa_t X_1 + (1 - \kappa_t) X_0$ subject to $X_1 \sim p_{\text{data}}$ and $X_0 \sim p_{\text{noise}}$. The interpolation schedule κ_t sets the probability of finding X_t in the denoised state X_1 at each time t . The interpolation schedule can be freely defined by the user so long as $\kappa_0 = 0$ and it monotonically increases to $\kappa_1 = 1$. Note that, for the MFMs analyzed in this section, this means that we can interpret κ_t as the CDF of the jump time for a position, $\kappa_t = \mathbb{P}(\tau < t)$. This, in turn, implies that the time derivative is the PDF of the jump time, $\dot{\kappa}_t = \mathbb{P}(\tau = t)$.

The CTMC underlying a DFM is used to sample transport paths to map samples from p_{noise} to final samples from p_{data} , the distribution the DFM seeks to model. The transport is simulated by sampling a sequence $X_0 \sim p_{\text{noise}}$ (all masked in the case of an MFM) and numerically integrating the CTMC by sampling

$$X_{t+\Delta t} \sim p_{t+\Delta t|t}(\cdot | X_t) \approx \delta_{X_t}(\cdot) + R(X_t, \cdot) \Delta t$$

until we reach $X_1 \sim p_{\text{data}}$.

The formulation of DFMs from Campbell et al.²⁵ and Gat et al.²⁶, which was described above, makes three important assumptions:

1. The conditional rates of the CTMC factorize per position, when conditioned on the denoised sequence X_1 .
2. The rates of multiple positions changing in the CTMC simultaneously is 0.
3. The interpolation schedule κ_t is independent of the partially denoised sequence X_t .

Given these conditions for the DFM, the rates for each position i can be found analytically to be

$$R_t^i(x_t^i, \bar{x}) = \frac{\dot{\kappa}_t}{1 - \kappa_t} \left(p_{1|t}(X_1^i = \bar{x} | X_t = x_t) - \delta_{x_t^i}(\bar{x}^i) \right), \quad (33)$$

where $p_{1|t}(X_1^i = \bar{x} | X_t = x_t)$ is the key object learned by a machine learning model. Note that this expression for the rates factors into two terms. The first is the probability of a transition occurring at time t , $\mathbb{P}_{\text{MFM}}(\tau = t | \tau \geq t) = \frac{\dot{\kappa}_t}{1 - \kappa_t}$. The second, $p_{1|t}(X_1^i = \bar{x} | X_t = x_t) - \delta_{x_t^i}(\bar{x}^i)$, ensures that, if there is a jump (when $\delta_{x_t^i}(\bar{x}^i) = 0$), the state is sampled from $p_{1|t}$. This is discussed in more detail in the proofs of the lemmas.

Similar to AO-ARMs, where we represent the sequence of partially unmasked states as the tuple (σ, X) , we represent the sequence of

transitions in a sampled MFM path as the tuple (τ, X) . Here, $\tau = (\tau_0 = 0, \tau_1, \dots, \tau_D < 1)$ is the sequence of transition times sampled by the CTMC, in increasing order, from 0 to 1. As before, $X = (X_0, X_{\tau_1}, \dots, X_{\tau_D})$ is the sequence of states at each of those times. Note that X_0 is the all-masked state while the fully unmasked state is $X_1 = X_D$.

As in the AO-ARM case, we say that σ is the permutation that maps from transition times to the position of the sequence that changed. Concretely, the i th transition occurs at time τ_i and causes the $\sigma(i)$ th position to be unmasked. Like AO-ARMs, this means that we can represent X as the tuple (σ, X_{τ_D}) . We can also use σ to sort the transition times of each position $T = (T_1, \dots, T_D)$ into the transition times of the CTMC: $\sigma(T) = \tau$.

Below, we include an expression for the probability of a CTMC path sampled using an MFM. This particular formula applies to all CTMCs and does not readily admit an algorithmic implementation; we include it here for completeness:

$$\begin{aligned} \mathbb{P}_{\text{MFM}}(\tau, X) \\ = \mathbb{P}_{\text{MFM}}(\tau_0, X_{\tau_0}) \times \prod_{i=1}^D \mathbb{P}_{\text{MFM}}(\tau_i, X_{\tau_i} | \tau_{i-1}, X_{\tau_{i-1}}) \times \mathbb{P}_{\text{MFM}}(\tau_{D+1} > 1 | \tau_D, X_{\tau_D}). \end{aligned}$$

MFM CTMCs use the minimal number of transitions. Campbell et al.²⁵ and Gat et al.²⁶ showed that the rates defined above (Eq. (33)) make the minimal number of jumps needed to denoise the sequence. In the case of an MFM, this means that we have exactly D transitions, one to unmask each position. However, there exists an infinite class of rate matrices that also solve the KFE and DFM boundary conditions, which do not have this minimal jumps property.

Nonminimal rate matrices can be formulated as those with nonzero stochasticity, as in Campbell et al.²⁵, and can be equivalently formulated through corrector sampling, as in Gat et al.²⁶. Our proofs explicitly handle the minimal case but the calculations can be readily extended to the case of nonzero stochasticity. However, we do note that, when performing exact sampling, rate matrices with nonzero stochasticity yield the same time marginals as the minimal rates (by construction). Generally, nonzero stochasticity is only useful when Δt is large enough during numerical integration that multiple positions are sampled at once. In such settings, stochasticity is believed to correspond to implicit ‘equilibration’ steps that reduce drift in the simulated CTMC.

Statement of exact MFM sampling using AO-ARMs. We show that paths from an MFM’s CTMC can be sampled exactly by sampling transition times uniformly at random and sampling state transitions from an AO-ARM independently. The key fact underlying our proof of Proposition 1 is that the transition times and states for an MFM are conditionally independent of each other given a decoding order. In particular, the distribution over paths factorizes into two terms. The jump time component reduces to a product of D independent uniform distributions, one for each position; the state transition component yields a product of denoising terms that correspond to the conditional distributions sampled by an AO-ARM. These preceding two statements are proven in Lemmas 1 and 2, respectively, below. Lastly, Corollary 1 extends the claim of Proposition 1 to show how to construct a sampler where the decoding order is sampled up front and the transition times and states are truly sampled independently. The detailed derivations and proofs are included in the Supplementary Text. Below, we state the main results.

We use this notation to show that the density of sampling τ and X from an MFM is the same as the probability of (1) sampling T uniformly at random, inducing an associated decoding order σ and τ as a result, and (2) using that decoding order to recover X from an AO-ARM, as described at the beginning of the proposition.

Together with Proposition 1, this completes our claim of conditional independence of the times and states given the unmasking

ordering. In Corollary 1, we can see plainly that the density of paths under the MFM CTMC is the same as sampling from the times and states independently once we have sampled the permutation up front. Furthermore, the corollary shows that, to get the likelihood of an output sequence under the CTMC's final distribution, the specific jump times can be marginalized out easily but the decoding order still makes getting likelihoods intractable.

Proposition 1. *The probability of sampling a path $(\tau, \mathbf{X})$ from an MFM is equivalent to the probability of*

1. *sampling transition times for each position in the sequence $\mathbf{T} = (T^1, \dots, T^D)$, uniformly at random, from the interval $[0, 1]$;*
2. *sorting $\mathbf{T}$ in increasing order to obtain a permutation σ , which maps the unmasking steps to the position to be unmasked at that step;*
3. *setting $\tau = \sigma(\mathbf{T})$, where σ acts as $\tau_i = T^{\sigma(i)}$;*
4. *sampling the sequence of states $\mathbf{X}$ from an AO-ARM using order σ with the MFM denoiser: $p(X^{\sigma(i)} = \tilde{x} | X^{\sigma(<i)} = x) := p_{1|t}(X_1 = \tilde{x} | X_t = x)$.*

Below, $\mathbb{P}_{OA}$ refers to the distribution induced by the resulting AO-ARM in point (4) above. We call this resultant model the 'equivalent' AO-ARM for our MFM. Note that we can also obtain the equivalent MFM for an AO-ARM by reversing the assignment of conditional distributions.

Altogether, these four points correspond to the following equation:

$$\mathbb{P}_{MFM}(\tau, \mathbf{X} = (\sigma, X_{\tau_D})) = U_{[0,1]}^D(\mathbf{T} = \sigma^{-1}(\tau)) \mathbb{P}_{OA}(X = X_{\tau_D} | \sigma),$$

where σ^{-1} acts as $T^j = \tau_{\sigma^{-1}(j)}$ and $U_{[0,1]}^D$ is the product of D independent uniform distributions on the interval $[0, 1]$.

Corollary 1. *An MFM can be exactly sampled by first sampling a permutation $\sigma \sim U[S_D]$ and then sampling the sequence of states for each transition from an AO-ARM. The transition times can then be sampled uniformly at random and assigned to positions according to the sampled permutation.*

Implications for guidance. Below, we use the equivalence between AO-ARM and MFMs to show a simplified proof of our guidance method in Nisonoff et al.¹⁷. The calculation performed here pertains specifically to MFMs but can be readily extended to DFMs more broadly. The core underlying intuition is that the jump times and state transitions depend on the current sequence only through $p_{1|t}^\theta$. This is because the interpolation schedule is independent of the state. Therefore, when using guidance, only this distribution needs to be modified.

When guiding an MFM, we condition the CTMC such that it samples from $p_{\text{data}}(x|y)$ instead of $p_{\text{data}}(x)$. This corresponds to setting $\mathbb{P}'(X_1) = \mathbb{P}_{MFM}(X_1|y)$ and solving for the appropriate rate matrices R' . Because of Proposition 1, we can instead analyze how to condition the equivalent AO-ARM and then push this change back into our MFM.

Proof sketch. The equivalent AO-ARM samples the unmasked values of positions using $p^\theta(x^{\sigma(i)} | x^{\sigma(<i)}) := p^\theta(X^{\sigma(i)} = x^{\sigma(i)} | X_t = x^{\sigma(<i)})$. To condition, we desire to sample from $p(x^{\sigma(i)} | x^{\sigma(<i)}, y)$ instead. All we require is an expression for this distribution using our existing conditional model, which we get using the Bayes rule as follows:

$$\begin{aligned} p^\theta(x^{\sigma(i)} | x^{\sigma(<i)}) &= p^\theta(x^{\sigma(i)} | x^{\sigma(<i)}, y) \\ &= \frac{p(y | x^{\sigma(i)}, x^{\sigma(<i)}) p^\theta(x^{\sigma(i)} | x^{\sigma(<i)})}{p(y | x^{\sigma(<i)})} \\ &= \frac{p(y | x^{\sigma(i)}, x^{\sigma(<i)})}{p(y | x^{\sigma(<i)})} p_{1|t}^\theta(X_1^i = x^{\sigma(i)} | X_t = x^{\sigma(<i)}) \end{aligned}$$

Above, $p(y | x^{\sigma(i)}, x^{\sigma(<i)})$ is identical to the noisy predictor trained in regular classifier guidance for an MFM. This suffices to recover the exact

guidance methodology in Nisonoff et al.¹⁷. Using the conditioned AO-ARM, we now write the equivalent MFM. There is one subtlety in this translation. Because we are now finding the general form of the rates absent an already sampled set of transition times and, therefore, decoding order, we must choose an appropriate order to use the guided AO-ARM's conditional distribution as the guided MFM's probability denoiser.

Say we are looking at the rate for the position j at a time leading up to the i th jump, such that $\tau_{i-1} < t \leq \tau_i$. We choose σ such that $j = \sigma(i)$ and $\sigma(<i)$ corresponds to the positions currently unmasked in x_t . With this construction, we find

$$\begin{aligned} R'_{t^j}(x_t^j, \tilde{x}^j) &= \frac{\dot{\kappa}_t}{1 - \kappa_t} (p_{1|t}^\theta(X_1^j = \tilde{x}^j | X_t = x_t) - \delta_{x_t^j}(\tilde{x}^j)) \\ &= \frac{\dot{\kappa}_t}{1 - \kappa_t} (p^\theta(X^{\sigma(i)} = \tilde{x}^j | X^{\sigma(<i)} = x_t) - \delta_{x_t^j}(\tilde{x}^j)) \\ &= \frac{\dot{\kappa}_t}{1 - \kappa_t} \left(\frac{p(y | X^{\sigma(i)} = \tilde{x}^j, X^{\sigma(<i)} = x_t)}{p(y | X^{\sigma(<i)} = x_t)} p_{1|t}^\theta(X_1^i = \tilde{x} | X_t = x_t) \right. \\ &\quad \left. - \delta_{x_t^j}(\tilde{x}^j) \right). \end{aligned}$$

In the case that $x_t^i = \tilde{x}^j$, $\frac{p(y | X_1^i = \tilde{x}, X_t = x_t)}{p(y | X_t = x_t)}$ simplifies to 1, and we can factor out the added ratio of conditional probabilities to get

$$\begin{aligned} &= \frac{\dot{\kappa}_t}{1 - \kappa_t} \frac{p(y | X^{\sigma(i)} = \tilde{x}^j, X^{\sigma(<i)} = x_t)}{p(y | X^{\sigma(<i)} = x_t)} \\ &\quad \times (p_{1|t}^\theta(X_1^i = \tilde{x} | X_t = x_t) - \delta_{x_t^j}(\tilde{x}^j)). \end{aligned}$$

Altogether, we have

$$R'_{t^j}(x_t^j, \tilde{x}^j) = R_t^j(x_t^j, \tilde{x}^j | y) = \frac{p(y | X^{\sigma(i)} = \tilde{x}^j, X^{\sigma(<i)} = x_t)}{p(y | X^{\sigma(<i)} = x_t)} R_t^j(x_t^j, \tilde{x}^j).$$

This result is exactly the one derived in Nisonoff et al.¹⁷ and has two important properties we point out below.

First, this conditional rate is zero for any transition involving the change of two or more positions, just like the unconditional rates. Therefore, exactly $D \times (S - 1) + 1$ rates need to be computed, where D is the number of positions, S is the size of the alphabet, $D \times (S - 1)$ are all the Hamming distance jumps for each position in the sequence and the extra '+1' at the end comes from the rate of the CTMC staying at the current state.

Second, this shows that we can sample from the guided AO-ARM to get exact predictor guidance and, therefore, only need to evaluate the classifier S times: once for each of the possible transitions at the position being decoded. When S is small enough, this lets us sidestep the need for the TAG, which was derived by Nisonoff et al.¹⁷. They developed TAG because of the inefficiency of evaluating a classifier $D \times (S - 1) + 1$ times to run exact guidance when sampling by numerical integration. Using AO-ARM sampling, we show exact guidance can be performed with only a constant dependence in the dimension and linear dependence in the alphabet size. Therefore, running TAG with the AO-ARM sampling approach only adds a constant time cost for using guidance.

Common setup across in silico experiments

We now provide an overview dedicated to each in silico experiment. As the main text presents a high-level overview, for each in silico experiment, we include full experimental details, including datasets, predictive models, baseline configurations and sampling settings for all methods, as well as additional results. Here, we first address several considerations that are shared across different experiments, including baseline configurations and hyperparameter selections.

Post hoc filtering. In all experiments, we compared to the generation from the pretrained model unguided, with and without post hoc filtering. To ensure a meaningful comparison, we allowed the pretrained model to generate unguided for the same wall-clock time as with guidance and then post hoc filtered to only the top k samples using the predictor, where k is the number of samples allocated to ProteinGuide. In other words, all methods were allowed to generate sequences for the same amount of time and they were all scored according to their top k sequences, as judged by the predictor.

Fine-tuning. Although this work centers on on-the-fly guidance and fine-tuning is not on the fly, we nonetheless included a comparison to fine-tuning as it is a widely used technique for adapting pretrained generative models. Note that fine-tuning is sometimes referred to as ‘supervised fine-tuning’ because one accesses labels to curate the set; however, as the procedure does not use the labels, we choose not to invoke this terminology.

For each task, we performed fine-tuning by continuing training the pretrained model under its original training objective on a task-specific set of selected sequences thought to most exhibit the desired property^{8,9,11,89}. When the pretrained models are very large (for example, ESM3 and ESM C), we performed fine-tuning with LoRA, a standard parameter-efficient fine-tuning method that injects low-rank trainable matrices into selected linear layers while freezing most pretrained weights⁵². Unless otherwise stated, we used a commonly used LoRA configuration with low-rank adapters of rank 16 (scaling factor = 32, dropout = 0.05) applied to the model’s attention transformations and the sequence output head, training only these added adapter parameters while keeping the pretrained weights frozen. These choices are consistent with prior large-scale LoRA studies that explicitly swept ranks including 16 and identified 0.05 as a useful dropout value in smaller-model regimes (the ~1 billion parameters of the public ESM3 model are comparatively low when compared to large language models where fine-tuning has been more extensively studied)^{52,90}. We optimized the adapters with a learning rate of 10^{-4} (no weight decay) using a linear warmup over the first 10% of updates followed by cosine decay. We found the number of necessary training steps to be task dependent and typically trained for at least 500–1,000 steps (with an effective batch size of 64–128) to ensure that the model is well trained. Checkpoints were typically selected on the basis of model loss on a small holdout set. We did not further tune these optimization hyperparameters beyond this fixed configuration; task-specific tuning and model selection were performed only through the experiment-specific data selection rule and holdout validation protocol, as detailed below. For smaller models (for example, ProteinMPNN), we performed full fine-tuning of all trainable parameters.

In practice, fine-tuning also requires the user to make several additional design choices beyond specifying the target property, including (1) which subset of sequences should constitute the fine-tuning set and (2) how to perform model selection and hyperparameter tuning, which typically necessitates holding out data for validation and choosing optimization settings (for example, learning rate schedules, weight decay, early stopping and the number of epochs) on the basis of some validation signal. Empirically, even in well-studied large-model settings, fine-tuning model selection is performed using held-out validation criteria and can exhibit rapid overfitting⁹; more broadly, fine-tuning performance can be sensitive to optimization hyperparameters and is commonly selected using validation set performance^{81,91–93}. Accordingly, for each experiment, we specified a task-appropriate data selection rule and validation protocol (described in experiment-specific sections) and used these held-out signals to tune hyperparameters for the fine-tuning baselines.

Guidance strength. The guidance strength γ in Eq. (16) is a hyperparameter that can affect the quality of generation. Conceptually, the

Bayes rule implies $\gamma = 1$ but one might desire to bias the generation more toward the conditioning signal and others have found $\gamma > 1$ to result in higher-quality samples for continuous-space diffusion models^{20,21}. In our experiments, we tested $\gamma \in \{1, 10, 100\}$ and did not systematically tune it further, although more extensive tuning might further improve generation. We list the final value of γ for each experiment below. In practice, we recommend the user to start with $\gamma = 1$ and monitor whether $p(y|x_t)$ increases over the generation trajectory. Depending on where the unconditional model started the generation trajectory and the distribution of $p(y|x_t)$ over the state spaces, $\gamma = 1$ might be not sufficient to increase the guidance signals. In such cases, one can increase the guidance strength to either $\gamma = 10$ or $\gamma = 100$ or perform more systematic tuning on the metrics on generation quality one cares about.

Sampling speed. For both unguided and guided generation, the primary factor affecting the speed of generation is the number of sampling steps taken. In AO-ARM sampling, the number of steps corresponds to the number of positions in the sequence. In flow-matching sampling with Euler integration, the number of sampling steps corresponds to the amount of discretization, with fewer steps (that is, larger Δt) typically resulting in faster generation at the cost of lower sample quality. For MLMs, before connections to discrete-space diffusion and flow models were made, practitioners use a variety of sampling methods. Some examples include directly decoding all the tokens from the fully masked sequence or decoding the sequence iteratively, with a commonly used heuristic to choose the decoding order, which prioritizes the token to be unmasked according to some confidence⁴⁷. As guidance can be performed with either ‘continuous-time’ (for example, Euler integration) or ‘discrete-time’ (for example, any-order autoregressive sampling) sampling methods (Fig. 1b), we used the number of sampling steps roughly corresponding to the number of steps taken by unguided generation (which might depend on the length of the generated protein).

In addition to the number of sampling steps, when performing guidance, the sampling speed is affected by whether exact or approximate guidance is used and the cost of performing forward passes on the predictor (and backward passes in the case of TAG). We also illustrate the effectiveness of our exact and approximate guidance methods under different settings. We used exact guidance for the more challenging multiproperty, Pareto extrapolative design and the *in vivo* experiment, where performing forward passes with the predictive model is also relatively cheap. We also used exact guidance for the enzyme guidance experiment, where we found TAG to struggle. In other experiments, we used TAG to show that, despite being approximate, it can perform quite well with high efficiency on a variety of tasks, including for fold-class-guided generation, where the large size of the ESM3 structure token alphabet (>4,000) makes exact guidance fairly inefficient.

Stability-guided inverse folding

We demonstrated how assay-labeled experimental data can be used to condition generative models for protein sequences, improving the success rates for such models in generating sequences with the desired function. To illustrate this use case, we guided ProteinMPNN with a large dataset of experimental folding stability measurements on miniproteins⁵³ to generate sequences with improved stability. Stability is reported in real-value units of $\Delta\Delta G$, which quantifies the difference in stability from the wild-type sequence. A protein is considered stable if it is at least as stable as the wild type ($\Delta\Delta G \leq 0$). To generate sequences with improved stability, we followed the experimental setup from Nisonoff et al.¹⁷. Specifically, we used ProteinMPNN¹ as the inverse-folding model (as opposed to the FMIF model used in Nisonoff et al.¹⁷). As described above, the AO-ARM nature of ProteinMPNN means that we can also treat the pretrained model as a masked flow-matching or diffusion model. Then, we apply guidance with the stability oracle

from Nisonoff et al.¹⁷ to obtain a stability-conditioned inverse-folding model. For full details on the dataset used, the training procedure of the stability oracle and the stability classifiers used for guidance, we refer the readers to Nisonoff et al.¹⁷ (Appendix F.4).

Setup and baselines. We applied this setup on eight randomly chosen proteins for which the generated sequences from ProteinMPNN are not already on average more stable than the wild type, some of which differ from those used in Nisonoff et al.¹⁷. For each generated sequence, we define success on the basis of two criteria: (1) the generated sequence should fold into the desired structure and (2) the generated sequence should be at least as stable as the wild-type sequence, $\Delta\Delta G \leq 0$. We used ESM3 (ref. 47) to predict the structure of the generated sequences and compare the r.m.s.d. between the predicted structure and the desired structure. We consider a sequence to achieve the desired fold if r.m.s.d. ≤ 2 Å following previous evaluation criteria²⁵. To evaluate the folding stability of our sequences, we use the $\Delta\Delta G$ regression model (that is, the stability oracle) from Nisonoff et al.¹⁷.

For each protein, we compared the generations from all methods on 100 sequences. Leveraging the equivalences between MFMs and AO-ARMs (described above), we performed two sets of comparisons. In the first comparison, following the setup in Nisonoff et al.¹⁷, where guidance was applied to FMIF, an inverse-folding model trained with an explicit masked flow-matching objective, we leveraged the equivalences between MFMs and AO-ARMs to guide ProteinMPNN as a flow-matching model with Euler integration (Fig. 2). In the second comparison, we used ProteinMPNN's default any-order autoregressive sampling (Supplementary Fig. 1). In both cases, we used TAG to approximate the guide-adjusted rates. For Euler integration, we used $\Delta t = 1/L$ where L is the length of the sequence. In contrast to autoregressive sampling, which requires L decoding steps, Euler integration can in principle use fewer than L steps, potentially reducing sampling time at the cost of discretization error. In this experiment, however, the proteins were short (44–72 aa) and the autoregressive sampler can leverage caching to make each decoding step efficient. As a result, autoregressive sampling was faster in absolute wall-clock time for both unguided and guided generation: generating 100 sequences took <2 s with autoregressive sampling, compared to 5–10 s with Euler integration. The relative overhead from guidance was smaller with Euler integration; averaged across proteins, ProteinGuide took approximately 1.5 times as long as unguided generation with Euler integration and approximately three times as long as autoregressive sampling. This larger relative overhead under autoregressive sampling reflects the fact that caching and the short sequence lengths reduce the cost of the pretrained model in each sampling step, making the additional predictor cost from guidance more prominent. All experiments were performed on a single NVIDIA RTX 6000 GPU.

To compare to fine-tuning, we followed a similar procedure to Widatalla et al.³⁷ by training ProteinMPNN further on its original objective on all the sequences more stable than the wild type in the stability dataset, comprising ~100,000 of ~669,000 sequences. As ProteinMPNN only has a modest number of trainable parameters, we performed full fine-tuning. To select hyperparameters for fine-tuning, we used the same validation split from Nisonoff et al.¹⁷ constructed by clustering examples by wild-type backbone and holding out 20% of backbones by cluster. We kept all other ProteinMPNN training settings consistent with the original recipe and tuned only the learning rate and weight decay of the AdamW optimizer⁹⁴, with learning rate in the range $\{10^{-4}, 5 \times 10^{-5}, 2 \times 10^{-5}, 10^{-5}, 5 \times 10^{-6}, 10^{-6}, 5 \times 10^{-7}\}$ and weight decay in the range $\{10^{-2}, 10^{-3}, 10^{-4}\}$. For each pair of learning rate and weight decay value, we trained on the training clusters and selected the checkpoint with the lowest validation negative log likelihood or perplexity on the held-out wild-type clusters. This protocol was designed to reduce overfitting to specific protein families and is possible because the stability dataset from Tsuboyama et al.⁵³ is very unique and contains the same

type of measurements across many diverse proteins (in contrast to single-family landscapes where protein-level holdout is not possible). The chosen settings included a learning rate of 10^{-6} and weight decay of 10^{-2} trained for around 25 epochs. After selecting hyperparameters on the validation split, we retrained the model using the selected hyperparameters on all clusters for the chosen stopping point to produce the final fine-tuning baseline used in evaluations.

For all methods, sampling was performed with a temperature of 0.1. Guidance was performed with TAG with a single guidance strength $\gamma = 100$. We did not specifically tune the sampling temperature or guidance strength.

Implementation note on decoding ordering in AO-ARMs. While the mathematical equivalence between AO-ARMs and MFMs is exact in theory, a practical subtlety arises when AO-ARMs are implemented with causal attention mechanisms. In practice, many AO-ARMs, including ProteinMPNN¹, are implemented using standard autoregressive architectures with causal attention masks. When computing the probability $p_\theta(x_i^j | x_t)$ for a masked position i given a partially masked sequence x_t , the model must process the unmasked positions in x_t in some specific order because of the causal nature of the attention mechanism. Formally, for a position $i \in M_t$ (the set of masked positions), the probability $p_\theta(x_i^j | x_t)$ should theoretically be independent of how the unmasked positions in $U_t = j : x_t^j \neq ?$ are ordered. However, in practice, the causal attention mechanism means that

$$p_\theta(x_i^j | x_t) = p_\theta(x_i^j | x_t^{\sigma(1)}, x_t^{\sigma(2)}, \dots, x_t^{\sigma(|U_t|)})$$

where σ is some permutation of the unmasked positions in U_t and the probability depends on this specific ordering σ . During training, the AO-ARM objective encourages the model to produce consistent predictions regardless of the permutation σ . The expectation over all possible permutations in the training objective should, in principle, ensure that $p_\theta(x_i^j | x_t)$ is invariant to the ordering of unmasked positions. However, in practice, this invariance may not be perfectly achieved because of imperfect optimization and finite-sample training. To address this subtlety and ensure consistent behavior during guided sampling, for the protein stability experiment that guides ProteinMPNN as a flow-matching model, we fix the decoding ordering for x_t on the basis of the residue positions of the unmasked tokens. Specifically, we sort the unmasked positions in U_t in ascending order of their sequence positions:

$$\sigma(j) = \text{sort}(U_t)[j]$$

where $\text{sort}(U_t)$ returns the unmasked positions ordered by their indices in the original sequence. This approach provides a deterministic and reproducible ordering for any given masked state x_t , respects the natural left-to-right order of residues in the protein sequence and requires no additional hyperparameters or complex ordering strategies. By using this fixed position-based ordering, we ensure that the conditional probabilities $p_\theta(x_i^j | x_t)$ are well defined and consistent across different sampling runs, enabling reliable application of the guidance method to AO-ARMs trained with causal attention.

Guiding ESM3 with assay-informed predictive models

For the extrapolative design task, we evaluated the ability of ProteinGuide to generate diverse proteins with property values that exceeded any seen in the supervised training data. We used assay-labeled datasets for three diverse proteins that include many variants with more than single substitutions: the CreiLOV fluorescent protein⁵⁴, UBE4B (ref. 55) and PABP⁵⁶ (Extended Data Fig. 4a), which have previously been used in other studies for evaluating protein fitness prediction and optimization^{34,42,43,95}. Below, we provide details for the dataset used, the oracle model used for evaluation, the setup for baselines and guidance and additional results.

CreiLOV. CreiLOV is a flavin-binding fluorescent protein derived from LOV photoreceptors; LOV fluorescent proteins were developed as genetically encoded reporters that work without molecular oxygen, making them attractive alternatives to GFP in hypoxic or anoxic (anaerobic) contexts. We used the dataset from the study of Chen et al.⁵⁴ processed by Yang et al.⁴², where higher fitness refers to brighter fluorescence. The dataset contains 167,530 variants, including (1) SSM covering all 118 residues (that is, all single substitutions) and (2) higher-order substitutions of 15 selected positions with beneficial single substitutions, with up to 15 substitutions from the wild type (Extended Data Fig. 4a, left).

UBE4B. UBE4B is an E3 ubiquitin ligase in the ubiquitin–proteasome system, where modulating ligase activity can change protein ubiquitination and downstream regulation; it is a useful engineering target because small sequence changes can strongly tune enzymatic activity. We used the assay-labeled dataset from Starita et al.⁵⁵, where fitness refers to E3 autoubiquitination activity (quantified by high-throughput selection or enrichment of variants with higher activity). Specifically, we used the version of the dataset processed by Hsu et al.⁹⁵, which includes 32,290 variants with substitutions of the 102 residues in the C-terminal region of the protein, with up to 6 substitutions from the wild type (Extended Data Fig. 4a, right).

PABP. PABP regulates mRNA fate and its RNA-recognition motifs (RRMs) are a classic system for studying and engineering RNA-binding function. We used the assay-labeled dataset from Melamed et al.⁵⁶, where fitness refers to *in vivo* functional complementation or growth-based enrichment of RRM2 variants (variants that better support cellular function are enriched) for the yeast version of PABP, usually referred to as Pab1. Specifically, we used the version of the dataset processed by Hsu et al.⁹⁵, which includes 37,711 variants with single and double substitutions in a 96-residue region of the protein (Extended Data Fig. 4a, middle).

Oracle models. Following a similar experimental setup to prior work^{34,42,43}, we used an oracle model trained on the full assay-labeled dataset as an *in silico* proxy to measure the fitness of the generated sequences. For CreiLOV, we used the published fitness oracle from Yang et al.⁴², which is an ensemble of ten two-layer MLP regressors trained on one-hot sequence encodings. Yang et al.⁴² used all of the single, double and triple mutants in the library for training, with 10% of the quadruple mutants for validation. The oracle achieved a Spearman correlation of 0.93 on its validation set (Extended Data Fig. 4b, left).

For UBE4B and PABP, we trained oracle models with similar architectures to above. Each model is an ensemble of five two-layer MLP regressors (hidden dimension size: 400) trained on one-hot encoded sequences. For each dataset, we trained the model on 90% of the sequences and validated on the remaining 10% sequences, with a learning rate of 10^{-3} (weight decay of 10^{-2}), using a linear warmup of 10% of updates. As the UBE4B and PABP datasets have fewer variants and lower-order mutants compared to the CreiLOV dataset, we did not specifically hold out higher-order mutants for testing so as to maximize the information the oracle could extract from the assay-labeled data. The oracle for PABP achieves a Spearman correlation of 0.92 on its validation set and the oracle for UBE4B achieves a Spearman correlation of 0.59 on its validation set (Extended Data Fig. 4b, middle and right).

Setup and baselines. We used ESM3 as the pretrained model. We compared guided generation to unguided generation and fine-tuning ESM3 with LoRA⁵². For sampling from ESM3, either unguided or after fine-tuning, we used the default temperature of 0.7 and L sampling steps, where L is the number of residues we are designing. As the predictor used for guidance is lightweight, guided generation runs at roughly the same speed as unguided generation (on average, it took

1–1.5 s to generate one sequence). Nevertheless, to be conservative, we generated 2.5 times as many samples for both unguided generation and fine-tuning when performing post hoc filtering. We performed generation starting from fully masked sequences so as not to bias the generation toward the wild-type sequence; however, note that conditioning on additional information, such as a partially masked wild-type sequence or predicted backbone structure, can help make the generation task easier.

For fine-tuning ESM3 with LoRA, we used the hyperparameter settings detailed above. For validation, we held out 100–200 sequences with the highest property value in the labeled dataset (roughly corresponding to the top 0.1% of CreiLOV dataset and the top 0.5% of the PABP and UBE4B datasets). We then experimented with performing fine-tuning on the sequences in the top $M \in \{2, 5, 10, 20, 50, 100\}$ percentile of the labeled dataset. For each hyperparameter setting, we trained the model with a minimum of 1,000 steps and an effective batch size of 128. We selected the threshold and checkpoint with the lowest perplexity on the holdout sequences to report generation results. We found that the perplexity was typically lowest for the models trained on the top 2–10%, for which the model typically converged after 1,000 training steps and the differences in holdout perplexities with the different thresholds were not significant.

Guidance. To perform guidance, we trained predictors directly on partially masked sequences to align with the guided sampling distribution. For each dataset, we used the same architecture to the oracle for the predictor used for guidance and trained them on 2,000 randomly sampled sequences from the labeled dataset to match the experimental budget in our later *in vivo* experiment, with the same set of variants held out as fine-tuning. Positions in each sequence were masked at random with a masking rate uniformly sampled in the range [0, 1]. We used the same architecture and hyperparameters as the oracle model for these predictive models. The predictors are very lightweight and took <5 min to train for each dataset.

Joint likelihood guidance. As ESM3 is a pan-protein model, we additionally guided sampling to remain within the desired protein family. We guided with a joint likelihood that combines (1) a property term encouraging high predicted property value and (2) an in-family term encouraging membership in the target family:

$$p(y \geq y^*, \text{fam} = 1 | x_t) = p(y \geq y^* | \text{fam} = 1, x_t) p(\text{fam} = 1 | x_t). \quad (34)$$

We parameterized $p(y \geq y^* | \text{fam} = 1, x_t)$ using an ensemble regressor; letting $\mu(x_t)$ and $\sigma(x_t)$ denote the mean and s.d. of the ensemble predictions, we defined

$$p(y \geq y^* | \text{fam} = 1, x_t) = 1 - \Phi\left(\frac{y^* - \mu(x_t)}{\sigma(x_t)}\right), \quad (35)$$

where $\Phi(\cdot)$ is the standard Gaussian cumulative distribution function (CDF). We parameterized $p(\text{fam} = 1 | x_t)$ with a Bernoulli likelihood whose probability is given by a binary classifier head. The classifier was trained to discriminate between (positive) sequences from the assay-labeled dataset and (negative) sequences sampled from unguided ESM3. For each protein, we trained the model on 500 sequences from ESM3's unguided generation, in addition to the 2,000 randomly sampled sequences from labeled dataset. Regression and classification losses were weighted equally.

Choosing the property threshold y^* . Although one could set $y^* = y_{\max}$ (the best observed labeled variant) to directly target extrapolation, we found that this yields a weak early guidance signal because predictors tend to output values near the training set mean when the sequence is heavily masked. To keep the guidance signal nondegenerate

throughout the trajectory, we set y' to be at or below the mean label value of the predictor training set (CreiLOV: 3.8, PABP: -1, UBE4B: -1.5). A natural extension is to anneal y' during sampling—starting near the training set mean early in decoding and increasing toward a more stringent threshold later—analogous in spirit to the progressively tightened conditioning distributions used in CbAS⁹⁶. While such annealing could plausibly improve extrapolation further, we found the fixed-threshold heuristic to be sufficient and did not explore annealing here.

Guided sampling. For guided generation, we used TAG with $\gamma = 10$ and $\Delta t = 0.01$, which corresponds to 100 integration steps, roughly corresponding to the design region lengths of the proteins in this experiment. We used the same temperature of 0.7 as with unguided generation. The joint likelihood provides two guidance terms that can compete early in generation, particularly before the partially unmasked sequence has entered the in-family region of ESM3's distribution. To stabilize generation, we optionally applied staged guidance; for PABP and UBE4B, we first guided using only the in-family term $p(\text{fam} = 1|x_t)$ and then switched to the full joint likelihood. Specifically, with $t = 0$ denoting a fully masked sequence, we began joint guidance at $t = 0.5$ for PABP and $t = 0.1$ for UBE4B and used joint guidance throughout the entire trajectory for CreiLOV. An alternative would be to use separate guidance strengths for the two terms; we did not pursue this to keep hyperparameters consistent across experiments.

Additional results. In addition to the results in the main text (Fig. 3), we report structure-based metrics computed from ESMFold predictions for generated sequences, including the distribution of predicted local distance difference test (pLDDT) and the refolded r.m.s.d. to the predicted wild-type structure (Supplementary Fig. 2). Unguided ESM3 generations frequently exhibit low pLDDT and large refolded r.m.s.d.; consequently, a fraction of guided samples also fall into this low-confidence range. Importantly, however, ProteinGuide also produces sequences that simultaneously achieve high predicted property values (Fig. 3d) remain diverse and different from the wild type (Fig. 3c) and exhibit favorable structure-based metrics, suggesting that the best guided generations are not merely exploiting the fitness oracle. In contrast, fine-tuned generations tend to remain very close to the wild-type sequence (Fig. 3c) and accordingly often yield higher pLDDT and lower refolded r.m.s.d.

Multiproperty guidance with ProteinGuide

Dataset. In this experiment, we used the data from the first library tested by Wang et al.⁵⁷. This library consisted of the wild-type sequence and 1,098 variants arising from an alanine scan of the protein, an SSM at 49 positions and a modest number of higher-order substitutions (up to a degree of 5) sampled from seven positions. Because the assay values saturate to finite values for nonfunctional variants, we use an inverse softplus transformation on the data and transform predictive model outputs back with a softplus such that they saturate at the same nonfunctional level as the real data.

Oracle and predictive models. Both the oracle and the predictive models in this experiment are regression models that use the kernel trick. We use a kernel engineered to allow meaningful use of mutational data even when the substitution being tested is not necessarily the same as the substitution observed in the data. Formally, for one-hot encoded variants X_1, X_2 and the one-hot encoded wild-type X_{WT} , the kernel for capturing first-order epistasis (substitutions in isolation) is $K_1(X_1, X_2) = (X_1 - X_{WT})^T (X_2 - X_{WT})$. Note that this regressor only weighs the effects of different substitutions through the variants, not at the level of individual epistatic terms. Because of this structure, higher-order kernels for the n th order simply need to count the n th-order terms that are shared between two variants. This is purely a function

of the first-order matches, that is, $K_n(X_1, X_2) = \binom{K_1(X_1, X_2)}{n}$. The predictive model is of the second order, $K_1 + K_2$, while the oracle is of the third order, $K_1 + K_2 + K_3$.

For a partially masked sequence x_t , a predictive model of Pb and Zn activity $f(x) = (\hat{y}_{Pb}, \hat{y}_{Zn})$, and a target region T , the noisy classifier was defined as $p(y|x_t) = \mathbb{E}_{p_\theta^*(x|x_t)}[\mathbb{1}[f(x) \in T]]$. Here, $p_\theta^*(x|x_t)$ is the product-of-marginals distribution for the pretrained generative model $p_\theta(x|x_t)$. We also estimate the expectation with 50 samples from p_θ^* .

Lastly, because of mismatches in the predictor and oracle, samples that look good under the predictor might fail when scored by the oracle. In a real design setting, there is no oracle access; hence, we aimed to somehow estimate this mismatch. We reasoned that, especially with our regression models that used the kernel trick, substitutions for outlier data points would cause extremely large fluctuations in the predicted values when seen in new mutational context. This is because the optimization problem when fitting the regression parameters is poorly conditioned. This can result in the regression coefficients for a few data points relevant for the prediction of the outlier being perfectly balanced against each other, with large values canceling each other out to match the true label. When a new variant is scored, a different subset of sequences may have high inner-product scores with it. Among this new subset, the unstable, large regression coefficients for an outlier sequence may suddenly dominate.

To quantify this bias for variants far from the data, we generated new variants with the non-SSM positions fixed and up to 30 substitutions selected uniformly at random. We then scored them with the trained predictor and additionally scored them with an ensemble of ten predictors, each fit on the training data with 20% of the examples randomly dropped out. We observed that Zn-binding affinity was typically underestimated by our model and Pb predictions tended to not have any obvious systematic bias (Extended Data Fig. 6e). Therefore, we adjusted the target region cutoffs when guiding for all experiments according to these residuals described above. Note that, for the guided fine-tuning model sequences, there was no systematic difference in the estimated values between the guidance predictive model and the oracle for either property of interest.

The oracle model was trained on the full dataset whereas the predictor was only trained on a subset of 1060 sequences that excluded some high performing and higher mutational depth sequences. The excluded sequences were used for the validation set (Extended Data Fig. 5a, left and right).

Experimental setup and baselines. For all models, we only designed the positions for which there were SSM data available. All results were compute-matched. The unguided model generated 1,000 sequences, the fine-tuned model generated 500 and ProteinGuide generated 100. All models were evaluated on their top 100 sequences. We used the exact guidance algorithm for ProteinGuide, described above.

The fine-tuned model was trained on the subset of predictor training data in the more Pareto-optimal arm of the experimental data distribution (Extended Data Fig. 5a, middle, points above the dashed line). It was trained until the loss converged. We did not observe any overfitting on training data. The validation loss converged at a similar time as well.

The original PbrR study fine-tuned a protein language model to iteratively redesign the best sequences in the dataset for the Pareto extrapolation task⁵⁷. They call this method multiobjective conservative extrapolation, which we refer to herein as MOCE. In addition to the baselines mentioned above, we compared to this method and found that it did not generate any successes (Extended Data Fig. 6d). This is likely because MOCE is designed to conservatively limit extrapolation and does not move far from the training data.

For MOCE, we used the training and generation scripts in the project GitHub repository. We trained the model on the same data as

the predictor because MOCE uses pairs of data to learn the directional impact of substitutions on sequences. We used the 30 sequences closest to the target region as the starting points for sampling. We also allowed MOCE to generate 600 sequences of which there were 299 unique variants and 100 of these were used for evaluation.

Enzyme class-guided sequence generation

In this experiment, we demonstrated how guidance can generate sequences from target Enzyme Commission numbers (ECs). ECs are hierarchical four-level codes describing different enzyme classes. We trained an enzyme class predictor that classifies protein sequences into their ECs on the basis of their sequence and used it to guide the protein language model ESM3 (ref. 47). We present results for guiding sequence generation toward EC numbers of varying prevalence in SwissProt⁹⁷. Our results focus on the top ten most common ECs (Extended Data Fig. 1c) but we also include data for ten random and rare EC classes (Supplementary Table 4).

Enzyme class predictor. Our enzyme class predictor was a three-layer multilayer perceptron (MLP) that took mean-pooled ESM3 sequence embeddings as input. The embeddings are of dimension 1,536 and we used an MLP with hidden dimension 3,072. The predictor was trained on the SwissProt dataset used by ProteinInfer⁹⁸. Our model was trained on all SwissProt sequences, including nonenzymes. Sequences for each EC were randomly divided into training and test sets, with the test set additionally including the test sequences for CLEAN⁹⁹, which we used for benchmarking our predictor. The test set contained sequences from ECs with at least ten samples. Those with at least 30 were used for training. Finally, 15% of the nonenzymes were included in the test set. The remainder were used as training data. The training data contained 315,833 samples and the test data contained 36,354 samples. The noisy predictor's dataset contained ten randomly sampled noising times for each sequence.

The model was trained with a standard negative log-likelihood objective and an AdamW optimizer (learning rate of 0.001, weight decay coefficient of 0.01). We used a batch size of 15,000 and trained for two epochs on the full training data. The accuracy curves for each mask-level bin are plotted over the test set of the 100 most common level-4 EC numbers (Extended Data Fig. 1b).

ProteinGuide setup and baselines. In this experiment, we implemented exact classifier guidance for discrete flow matching as described in Nisonoff et al.¹⁷ using the exact sampling algorithm described above. As mentioned, this is similar to the Gillespie algorithm described in Peng et al.⁸⁰. The key difference is that their Gillespie algorithm is for unconditional discrete diffusion, whereas our proof gives a general method for converting any CTMC with state-independent and position-wise-independent transition probabilities. We used a guidance strength $\gamma = 10$ on the classifier outputs and ESM3 sampling temperature of 0.7. We sampled all positions one at a time.

This experiment included three baselines: post hoc filtering, a LoRA fine-tuned model and ESM3 using natural language prompting. For post hoc filtering, we took 210 unconditional samples for each enzyme class such that the comparison was compute-matched between the two models. Exact guidance requires 21 unconditional model evaluations and the cost of the classifier ends up being negligible by comparison. We fine-tuned ESM3 using LoRA according to the setup described above. A separate LoRA model was trained for each EC. Because of the additional training cost of fine-tuning of this experiment, we only allowed it the same sampling budget as guidance. We used an 80:20 split between training and testing. We trained each LoRA model to convergence and took the checkpoint with the lowest test loss (average cross-entropy for masked positions). For natural language prompting, we curated a set of the keywords from the ESM3 IntroPro keyword vocabulary that were most similar to the official EC

name using fuzzy string matching (full list in Supplementary Table 3). We then sampled from ESM3 with these keyword prompts as additional conditioning information during sequence generation.

Evaluation metrics. We considered the generated sequences a success if (1) the predicted class with the highest probability was the target EC and (2) ESMFold predicted a median pLDDT across residues of at least 0.8. The oracle model for this experiment was trained the same way as the predictor using SwissProt.

Detailed results. In our main experiment, we had each method (unguided with and without post hoc filtering, fine-tuning, natural language prompting and ProteinGuide) generate ten sequences for each of the ten most common enzyme classes (Extended Data Fig. 1). We did not observe a significant difference in the success rate between the fine-tuned model and ProteinGuide, as was expected for an interpolation task. We also did not observe a significant difference in the average Hamming distance between sequences generated by ProteinGuide or fine-tuning and the closest wild type in SwissProt (Supplementary Table 5). We did not observe any successes from the unguided model or the natural language prompting. We further found that there was not much difference in average Hamming distance between the guided and unguided sequences (Supplementary Table 6).

We additionally ran an experiment where we compared the ability of unguided sampling and ProteinGuide to redesign partially masked enzyme sequences to recover another sequence of the same EC (Extended Data Fig. 1d). We used several masking levels (50%, 60%, 70%, 80%, 90% and 100%). We found that ProteinGuide displays a consistent improvement over the unguided model. At high masking levels, this is a substantial relative improvement over the unguided model but a small relative improvement at low masking levels. It is worth noting that we found no detectable degradation in the pLDDTs of designed sequences because of the guidance process.

Lastly, we report results for ten randomly selected and the ten rarest ECs (Supplementary Table 4). For these experiments, we started generation from real enzyme sequences with 80% of positions masked. We found that ProteinGuide performed similarly on the top ten classes and ten random classes. However, the ten least represented classes were not in the training data for the predictor; accordingly, accuracy suffered. ProteinGuide was still able to achieve success about one third of the time. We believe that this was because of the predictor's cross-entropy training objective, allowing it to guide to the target classes essentially through a 'process of elimination', in that guidance is really just avoiding the classes for which the predictor does have trained logits.

Fold-class-guided backbone structure generation

We first provide an overview of the experimental setup for guiding ESM3 backbone structure generation with fold class labels from CATH¹⁰⁰, a hierarchical protein structure classification scheme. Concretely, we trained a fold classifier, $p_\phi(\mathbf{c}|\mathbf{x})$, that predicts the CATH labels, $\mathbf{c}$, from the discrete structure tokens outputted by the structure encoder of ESM3, $\mathbf{x}$, on the set of CATH-labeled protein structures from the Protein Data Bank (PDB). We combined this fold class predictor with the structure generation track of ESM3, $p_\theta(\mathbf{x})$, to obtain a fold-class-conditioned backbone structure generative model, $p_{\theta,\phi}(\mathbf{x}|\mathbf{c})$ (Extended Data Fig. 2a). We designed this experiment to demonstrate that ProteinGuide can be used to guide a large multimodal protein language model to generate structures.

Fold classifier. Now, we describe additional details of the fold class oracle and the fold classifier used in guidance. As ESM3 operates on discrete structure tokens, we need a predictor that predicts the fold class label of a protein from its structure token representation. To this end, we first trained a fold class predictor that takes in discrete structure

tokens as inputs. The model architecture consists of an encoder-only transformer trunk. The final per-position embedding is mean-pooled and then fed into three parallel linear layers, each projecting the embedding to a vector with the dimension equal to the number of classes for each of the three levels of CATH hierarchies: class, architecture and topology (fold). The C level describes the overall secondary-structure content of a structure (for example, 'mainly α ', 'mainly β ' and 'mixed α/β '), the A level roughly clusters structures by the overall arrangement of secondary structure in space (for example, ' β -barrel' and 'roll') and the T level assigns structures by how the secondary-structure elements are connected and arranged (for example, 'immunoglobulin-like' and 'TIM barrel')¹⁰⁰. The model was trained on the PDB dataset following the same preprocessing procedure as in Geffner et al.¹⁰¹ with a total of 214,564 structures, categorized into five C classes, 43 A classes, and 1,336 T classes. We fed each structure through the structure encoder of ESM3 to obtain its structure token representation. The dataset was randomly divided into training, validation and test sets at a ratio of 8:1:1, ensuring that at least one protein from each class was included in the test set whenever possible.

The model was trained with a standard cross-entropy loss applied to each of the CATH levels. Some proteins have multiple CATH labels because of having multiple domains and CATH being a domain-level annotation. For such proteins, we followed Geffner et al.¹⁰¹ and randomly sampled one domain label as the ground truth for each training iteration, encouraging the model to predict equal probabilities for all domain labels of a protein. During evaluation, if the model predicted any of the correct labels, the prediction was considered correct. For each protein, the loss was calculated for each level of its available CATH label. The transformer trunk of the model had a hidden dimension size of 512, 16 attention heads and five encoder layers. In total, the model had around 19 million parameters. The model was trained using the AdamW optimizer^{94,102} with a learning rate of 0.0001, weight decay coefficient of 0.01, dropout rate of 0.2, batch size of 128 and gradient accumulation step of 4. The model was trained for around 26,000 training steps on a single NVIDIA RTX 6000 GPU. We did not systematically tune the hyperparameters of the model. On its test set, the model achieved an aggregated micro accuracy of 0.9678 (A level) and 0.9490 (T level) and macro accuracy of 0.9702 (A level) and 0.9212 (T level)¹⁰³.

The fold classifier used for guidance takes in partially masked inputs. We initialized the model weights of this 'noisy' classifier with those of the unnoised classifier and trained it on the same training dataset as the unnoised classifier. We trained the noisy classifier for around 28,000 additional training steps, with the same hyperparameters as the unnoised classifier.

Setup and baselines. We used ESM3 as the pretrained model (with its structure token track) and performed fold class conditioning on the A and T levels of CATH, providing a practically informative middle ground between the coarser C level, which is often too broad to be challenging or discriminative, and the finer-grained H level, which is highly specific and can be overly restrictive for controlled generation. We compared the methods on the ten most common classes and ten randomly selected classes for both A and T level (Supplementary Table 8), evaluating on $k = 100$ samples for each class. The generated structures were evaluated on two criteria: (1) whether the fold class oracle predicts the desired class as the most likely class for the generated structure and (2) whether the structures are considered designable, where designability is calculated following standard procedure¹⁰⁴. For each generated backbone, eight sequences were sampled using ProteinMPNN¹ with a temperature of 0.1 and then ESMFold⁷¹ was used to predict the structure of each generated sequence. The r.m.s.d. was calculated between the generated structure and the predicted structure and a generated backbone was considered designable if the lowest r.m.s.d.—referred to as the self-consistency r.m.s.d.—among the eight sampled sequences was ≤ 2 Å.

We compared ProteinGuide to unguided ESM3 sampling (with and without post hoc filtering). We additionally compared ProteinGuide to ESM3's built-in prompt conditioning, which was trained to accept keyword-style 'function tokens' as conditioning signals. Following the procedure in Hayes et al.⁴⁷ and the ESM3 GitHub repository, we converted each target CATH label to a set of ESM3-compatible keywords using an intermediate mapping through InterPro. Specifically, for architecture (two-part codes) and topology (three-part codes), we first mapped the CATH code to one or more associated InterPro entries and then converted each InterPro identifier to a (possibly multikeyword) list using ESM3's provided InterPro-to-keyword dictionary (for example, tokens such as β , barrel and helix). Each keyword was provided to ESM3 as a function annotation spanning the full sequence (positions 1 through the target length); when multiple InterPro entries mapped to multiple keywords, we supplied the union of the resulting annotations.

As for the comparison to fine-tuning, we performed LoRA-based fine-tuning of ESM3 on proteins from the desired classes. Because the targets here are discrete labels, we performed fine-tuning separately for each class, substantially increasing the overall time and compute cost for fine-tuning compared to guidance. We used the same LoRA setting as previously described, with the differences that we fine-tuned the structure output head instead of the sequence output head. For each class, we performed fine-tuning on 90% of the examples, with 10% holdout for validation, training for a minimum of 500 steps and a maximum of 5,000 steps, with an effective batch size of 64 and early stopping imposed if validation loss stopped decreasing for more than five epochs. We found setting a sufficiently large minimum number of training steps to be important for fine-tuning, especially for classes with a smaller number of examples. Depending on the number of samples and how quickly they converged, fine-tuning on a single class can take between several hours on small classes to more than half a day on the larger classes on a single GPU. Because of the additional training cost of fine-tuning of this experiment, we only allowed it the same sampling budget as guidance.

Following evaluation procedures in prior work¹⁰¹, for each class and for every sample, we sampled its length uniformly among the lengths present for the proteins of that class in the AlphaFold Database, with a minimum length of 50 and maximum length of 275. For unguided generation, keyword prompting and fine-tuning, we used the default sampling setting of ESM3 with a sampling temperature of 0.7 and L sampling steps, where L is the length of the protein sequence¹⁰¹. We also experimented with using a lower sampling temperature and found that it results in ESM3 producing predominantly mainly α proteins. For guided generation, we used a sampling temperature of 0.3 and guidance strength $\gamma = 100$. We used Euler integration with TAG with a step size $\Delta t = 0.005$. We found using a sampling stochasticity $\eta = 20$ to improve guided generation but not unguided generation. ProteinGuide took approximately 1.2 times as long as unguided generation.

Detailed results. As the goal of this task is to generate more samples similar to those observed in the training distribution and because fine-tuning separately for each class allows each fine-tuned model to fully concentrate its distribution to the target class, we expected fine-tuning to be a very strong baseline. Indeed, we observed that fine-tuning performed quite well across different levels and classes (Extended Data Fig. 2b–d). The ProteinGuide generations were also substantially enriched for the desired annotations compared to unguided generation and keyword conditioning, with success rates comparable to the strong baseline of class-specific fine-tuning (Extended Data Fig. 2b–d). We observed keyword conditioning to be helpful for some classes but not the majority of cases. Furthermore, as expected, generation for more rare classes was more challenging (Extended Data Fig. 2c,d). We also observed that, across all generation methods, among the samples that were classified correctly, only a modest fraction were designable. This aligns with previous observations

that ESM3 tends to produce backbones that are less designable, possibly because of the composition of its training set¹⁰¹. We expect that the quality of generated structures should improve concomitantly with improved backbone generative models, independent of guidance. Examples of the generated backbone structures by guiding ESM3 with some example of CATH classes are shown in Extended Data Fig. 2e.

In vivo experiment: guiding base editor for improved editing activity

Base editors are engineered CRISPR effectors that can make transition mutations in the DNA by directly deaminating cytosine or adenine in the single-stranded DNA exposed during target recognition by Cas9. We focused on ABEs because of their importance as an engineering target, their small size and the availability of existing methods for measuring activity based on reversion of an inactivated antibiotic resistance^{59,62} that are suitable for high-throughput screening and detection. Additionally, ABEs (EC 3.5.4.33) are an example of deaminases, a large class of enzymes with important biomedical and biotechnological applications, including other base editors, RNA deaminases and purine and pyrimidine deaminases^{105,106}.

We explored how experimental data for base editor activity could be used to condition pretrained generative models to produce sequences with improved activity. As an overview, we first designed a first-round library with pretrained generative models. We collected experimental assay-labeled data on each of the designed sequences measuring their activities. Then, we trained a predictor for activity on the basis of the first-round sequences, which we then used to guide the design of a second-round library with ProteinGuide. Our goal was to generate sequences in the second round with improved activities compared to those in the first round.

ABE library cloning. ABE libraries contained a diversified region of 86 aa (positions 82–167) that was synthetically generated as an oligo pool and purchased from Twist Bioscience. The pool was amplified with KAPA polymerase (KAPA HiFi HotStart polymerase, Roche) using the following conditions: 1× KAPA HiFi 2× ReadyMix, 300 nM forward and reverse primer, 5 ng of template DNA and 3 min of initial denaturation at 95 °C, followed by 20 cycles of 98 °C for 20 s, 64 °C for 15 s and 72 °C for 1 min, with a final extension for 2 min at 72 °C. The resulting product was loaded on 2% agarose gel and gel-extracted using the DNA cleanup and concentration kit (Zymo Research). This pool was then cloned into a vector carrying the rest of the Tada sequence fused to dSpRYCas9 with an XTEN linker following the architecture of ABE0.1 (Supplementary Table 9)⁵⁹. The library was cloned by Golden Gate assembly using BsaI restriction enzyme and T7 ligase (New England Biolabs). The library (0.45 pmol in total) was mixed with 0.15 pmol of vector and cycled for 5 min at 37 °C and 5 min at 16 °C for 30 cycles, followed by 30 min at 37 °C and 5 min at 65 °C. The assembled library was purified using the DNA cleanup and concentration kit (Zymo Research) and transformed in One Shot Top10 competent cells (Thermo Fisher Scientific) by electroporation. Next, 950 µl of SOC medium was added and cells were incubated at 37 °C in a shaking incubator at 200 rpm for 1 h. Samples were taken for titer plates to estimate transformation efficiency and the remainder of the recovery was inoculated in 2×YT medium supplemented with 36 µg ml⁻¹ chloramphenicol and grown for 8 h. Plasmid DNA was purified with a QIAprep Spin miniprep kit (Qiagen) and used for selection experiments.

Base-editing selection assays. To measure adenine base editing, we generated a reporter carrying a carbenicillin resistance gene with a stop codon mutation (Supplementary Table 10) and a corresponding guide RNA and kanamycin resistance gene for plasmid maintenance. The selection strain was produced by transforming Top10 *E. coli* cells with the reporter plasmid and preparing electrocompetent cells. Briefly, the plasmid was transformed in One Shot Top10 (Thermo

Fisher Scientific) cells and a single colony was inoculated in liquid culture with 2×YT medium and 50 µg ml⁻¹ kanamycin. Cells were grown to mid-log phase and spun down at 3,500g, washed twice with ice-cold MiliQ water and twice with 10% glycerol (v/v). Each spin was performed for 10 min at 3,500g and, after a final spin of 13 min, the cell pellet was resuspended with 2 ml of 10% glycerol for 1L of starting culture, aliquoted, flash-frozen in liquid nitrogen and stored at –80 °C. For the selection experiments, 100 µl of cells were transformed with 200 ng of library plasmid and electroporated; then, 900 µl of SOC medium was added. After 1 h of recovery, the cells were inoculated in 4 ml of 2×YT medium supplemented with 50 µg ml⁻¹ kanamycin (nonselective antibiotic for the reporter plasmid), 36 µg ml⁻¹ chloramphenicol (for the base editor plasmid) and 0.1% arabinose for induction of the base editor and cells were induced for 20 h (for library 1 experiments) and 16 h (for library 2). After that, cells were washed twice with 2×YT to remove inducer and split in two. Half of the cells were inoculated in 30 ml of 2×YT with 50 µg ml⁻¹ kanamycin and 36 µg ml⁻¹ chloramphenicol (nonselective) and the other half were inoculated in 30 ml of 2×YT with 400 µg ml⁻¹ carbenicillin and 36 µg ml⁻¹ chloramphenicol (selective). Cells were grown to mid to late log phase and spun down; plasmid DNA was isolated using a QIAprep Spin miniprep kit (Qiagen). Next, 100 ng of plasmid DNA was used as the template for a PCR1 reaction amplifying the variable base editor region and adding adaptors for Illumina sequencing (Supplementary Table 7). The base editor portion was amplified using Q5 polymerase as follows: 100 ng of plasmid DNA as input, 200 nM forward and reverse primer and 1× Q5 MasterMix. The reaction was carried out at 98 °C for 30 s, followed by 20 cycles of 98 °C for 10 s, 63 °C for 10 s and 72 °C for 1 min, with a final extension of 5 min at 72 °C. DNA was purified with the DNA cleanup and concentration kit (Zymo Research) and sent to the Innovative Genomics Institute next-generation sequencing (NGS) core facility for PCR2 amplification and sequencing. The library was sequenced with NextSeq 1000/2000 P1 reagents (300 cycles) or MiSeq reagent kit v2 (600 cycles). Selection experiments for the comparisons to ABE1.1, ABE7.10 and ABE8 were performed in the same selection strain as the library experiments. For the titer plate assays, 25 µl of competent cells were transformed with 50 ng of each plasmid, cells were electroporated and incubated at 37 °C for recovery in 475 µl of SOC. They were added to 2 ml of 2×YT medium supplemented with 50 µg ml⁻¹ kanamycin, 36 µg ml⁻¹ chloramphenicol and 0.1% arabinose and induced for 6 h. Serial tenfold dilutions were made on selective (400 µg ml⁻¹ carbenicillin and 36 µg ml⁻¹ chloramphenicol) and nonselective antibiotics (50 µg ml⁻¹ kanamycin and 36 µg ml⁻¹ chloramphenicol) and the ratio of colonies on the selective over nonselective condition was calculated. For the pooled experiment with ABE1.1, ABE7.10 and ABE8 all plasmids were mixed in a 1:1 ratio and 100 ng of the plasmid mix was transformed in the selection strain. Transformation and induction were performed identically to the library experiments except that cells were induced for 6 h. After the selection, the plasmid DNA was extracted and the ABE region was amplified by PCR and sent for nanopore sequencing. Next, log enrichment was calculated identically to the library experiments.

Processing sequencing data. The paired-end 2×150-bp sequencing reads exhibited variable-length overlapping regions resulting from additional nucleotides incorporated between the Illumina adaptors and the primer annealing regions. To process and merge these reads, we first aligned the paired reads to the primer-binding sites at the 5' and 3' termini. In the event of nucleotide call disagreement between the forward and reverse reads in the overlapping region of the two reads, the nucleotide call with the higher Phred score was retained. These merged reads were then matched against the designed library of DNA sequences, keeping only reads with exact matches. To calculate log enrichments between the selective and nonselective conditions,

we added a pseudocount of 1 to all read counts for both conditions. The log enrichment, $\log e_i$, for each library member indexed by i was computed as

$$\log e_i = \log \left(\frac{n_i^s / N^s}{n_i^{ns} / N^{ns}} \right), \quad (36)$$

where n_i^s and n_i^{ns} represent the read counts for library member i in the selective and nonselective conditions, respectively, and N^s and N^{ns} denote the total number of read counts in the selective and nonselective conditions, respectively. We further filtered out any variant with counts ≤ 10 in the nonselective condition as the log enrichment estimates of these variants can be especially noisy.

Pretrained sequence generative models. We hypothesized that the structure of the wild-type TadA protein contains important constraints for maintaining a baseline editing activity, from which we could seek to improve. Therefore, we chose to use sequence generative models that we could explicitly condition on the wild-type TadA backbone structure. Specifically, we explored the use of two different pretrained sequence generative models, ESM3 and FMIF, an inverse-folding model that we trained with very similar training data and architecture to ProteinMPNN but with the explicit flow-matching training objective. We trained the FMIF model using the same dataset and training protocol described in Nisonoff et al.¹⁷ with two exceptions: (1) we used three encoder and decoder layers instead of four, as both hyperparameter settings performed equally well on a held-out test set and (2) we removed all examples of engineered ABE variants from the 2021 multichain training dataset provided by Dauparas et al.¹. This was performed to ensure that FMIF does not leverage helpful information for engineering ABEs that is generally not available when engineering a new protein. Specifically, we removed PDB 6VPC chains E and F from the training set. For ESM3, we used it as an inverse-folding model by providing the backbone structure as conditioning inputs to the model when generating sequences.

First-round library design. We built the first-round library by designing two sets of sequences with ESM3 and FMIF separately and then combined the sequences into a pooled library. The wild-type TadA is a homodimer and we sought to adhere to this constraint when generating the sequences. To this end, at each sampling step, we obtained the output logits of the models when conditioning on the structure of homodimer. That is, if the protein was of length L , the output logits were of size $2L \times S$, where S is the alphabet size of the model. We then took the average between the logits, which corresponds to the same position on the sequence of the monomer following Dauparas et al.¹. We used the averaged logits to parameterize the rate matrix used for sampling. As a result, at each sampling step, we only generated L residues, one for each position on the monomer, while respecting the structural constraints of the homodimer.

In many practical protein engineering applications, it is often desirable to have a control over the number of substitutions introduced to the wild type. In our case, we hoped to have a range of substitution counts in the first-round library, as we did not know a priori how tolerant TadA is to substitutions. To this end, we used an additional sampling hyperparameter used in ProteinMPNN¹, which we called the wild-type weight, w . At each sampling step, the output logit of the model was biased by the wild-type weight with $l_{ds}^w = l_{ds} + w$ for every position d of the sequence, where l_{ds} is the original output logit of the model for position d and state s , with s being the state of the wild-type sequence at position d . Setting $w = 0$ recovers sampling from the original model and increasing w allows one to gradually decrease the number of substitutions made to the wild-type sequence, where $w \rightarrow \infty$ results in only the wild-type sequence being sampled. The biasing of the logits by the wild-type weight can be interpreted as sampling from a weighted

mixture between the original distribution parameterized by the model and a delta distribution centered on the wild-type sequence. We note that, in principle, constraining the number of substitutions can also be imposed by introducing a guidance term that encourages the desired number of substitutions. However, doing so in a computationally tractable way might be challenging; thus, we leave this investigation to future work.

In our first-round library, we designed around 1,000 sequences with ESM3 and FMIF, for a total library size of around 2,000 sequences. In addition, we kept three copies of the wild-type TadA sequences in the library as controls. We chose to only design the region of 86 residues on the C-terminal region of the TadA protein because of the cost constraints of gene synthesis and we hypothesized that this region might be enriched for substitutions important for increasing editing activity according to prior studies^{59,61,62}. We fixed active-site residues P86, C87 and C90 and allowed the model to design the rest of the region. For both ESM3 and FMIF, we used eight different values of wild-type weight w linearly spaced between 0 and 3.5. We sampled from both ESM3 and FMIF using Euler integration with a time step $\Delta t = 0.002$, with a temperature of 0.5 and sampling stochasticity $\eta = 20$.

Predicting ABE activity from the first-round library. The log enrichment data collected from the first round of designs was used to design a second library enriched in highly active base editors. Specifically, we sought to generate sequences with an average first-round log enrichment score of at least 0.0 (Extended Data Fig. 7a). To do this, we trained a classifier to predict, given a sequence, whether the sequence's log enrichment is greater than zero. Guidance requires a classifier trained not only on 'clean' or 'unnoised' sequence-label pairs, $\{x_i^c, y_i^c\}$ but also on 'noised' sequence-label pairs $\{x_i^d, y_i^d\}$, $t \in [0, 1]$. This was achieved in two steps; first, a classifier was trained on just the unnoised data and then the classifier was further trained on the noised data.

First, the data were preprocessed by discarding sequences with highly variable log enrichments between the two replicates. Specifically, any sequences for which the absolute value of the difference in log enrichments between the two replicates exceeded 0.25 were discarded. This step filtered 114 of 2,000 sequences. Next, to further mitigate the influence of experimental noise, we discarded sequences with average log enrichment scores very close to the class boundary. Specifically, any sequences for which the average log enrichment was between -0.25 and 0.25 were discarded. This step filtered an additional 348 sequences, resulting in a dataset of 132 positive sequences (that is, mean log enrichment ≥ 0.25) and 1,406 negative sequences (that is, mean log enrichment ≤ -0.25).

Given the relatively small dataset size and notable class imbalance, we implemented a linear classifier augmented with pairwise interaction terms. This approach draws inspiration from Potts models, which are exponential family models that effectively capture pairwise dependencies between positions and have demonstrated robust empirical performance in protein sequence modeling¹⁰⁷. We trained the model using PyTorch^{108,109}, minimizing the binary cross-entropy loss with the AdamW optimizer^{94,102} with a regularization parameter of 10.0 applied specifically to the pairwise interaction terms. Training proceeded for 400 epochs using full-batch gradient descent until convergence of the loss function was observed. Across ten class-stratified 80:20 train-validation splits, the model achieved an average validation area under the receiver operating characteristic curve of 0.92.

Next, to adapt the classifier to also predict class labels for noisy data, we froze all single-site terms involving nonmask tokens and any pairwise interaction terms between two nonmask tokens (that is, we only further trained single-site parameters corresponding to mask tokens and trained pairwise interaction terms involving at least one mask token). For each gradient step, we first sampled the entire unnoised dataset; then, for each sequence in the dataset, we randomly sampled a time between 0 and 1 and masked the sequence by sampling

from the forward noising process up to that time. The model was again trained for 400 epochs without any regularization, at which point the loss function converged. Training the model in this way, wherein the parameters for the nonmask terms are frozen, has the nice property that the noisy model is guaranteed to be equivalent to the nonnoisy model at $t = 1$.

Second-round guided ABE library design. We generated the second-round library by guiding ESM3 and FMIF with the classifier we trained on the basis of the experimental data collected on sequences from the first round. We used the same sampling hyperparameters for each model as we did in the first round. We used exact guidance with guidance strength $\gamma = 100$. We designed 944 sequences with each model. All of the generated sequences had a predicted probability of being highly active > 0.5 . To enable direct comparison between the first-round and second-round libraries, we also included 112 sequences from the first-round library as controls in the second-round library so that we could calibrate the log enrichment measurements between the two rounds (Supplementary Fig. 3a). These 112 control sequences were selected to span the full range of observed log enrichment in the first round. This design allowed us to quantify and correct for batch effects between the two experimental rounds.

The 1,888 designed sequences and 112 control sequences were assayed and sequenced with the same protocols as in the first round (Extended Data Fig. 7b). We then used the control sequences to learn a linear transformation that maps log enrichment values from the second round to their corresponding values in the first round. Specifically, we fit a linear regression model:

$$\log e_{\text{round1}} = \alpha \cdot \log e_{\text{round2}} + \beta \quad (37)$$

where $\log e_{\text{round1}}$ and $\log e_{\text{round2}}$ represent the log enrichment values for the same sequence in rounds 1 and 2, respectively, and α (slope) and β (intercept) are the parameters of the linear transformation. The fit was performed on 109 sequences after filtering out sequences with a preselection count ≤ 10 in any replicate.

To assess the uncertainty in this calibration procedure, we implemented a bootstrap approach using the control sequences. Specifically, we performed 1,000 bootstrap iterations, each time randomly sampling 109 sequences with replacement from our control set and fitting a linear regression model. For each bootstrap sample, we derived a unique set of mapping parameters (α_i, β_i) and used these to transform the entire second-round dataset. We then calculated the median log enrichment value of the transformed second-round sequences for each bootstrap iteration, generating a distribution of potential outcomes that reflects the inherent variability in our experimental measurements. From this distribution of bootstrap outcomes, we identified the 5th–95th percentile mappings on the basis of the median log enrichment values. These percentiles represent conservative and optimistic estimates, respectively, of the transformation from round 2 to round 1 log enrichment values (Supplementary Fig. 3b).

Nucleic acids and plasmid preparation. All DNA oligonucleotides used in this study were synthesized by Integrated DNA Technologies. Plasmids were generated using Gibson assembly.

Reporting summary

Further information on research design is available in the Nature Portfolio Reporting Summary linked to this article.

Data availability

Data associated with the code and analyses are available on Zenodo (<https://doi.org/10.5281/zenodo.20496526>)¹¹⁰. Raw sequencing data for the base editor experiment are available from the National Center for Biotechnology Information through BioProject [PRJNA1467757](https://www.ncbi.nlm.nih.gov/bioproject/PRJNA1467757).

Primers and protein sequences are available in the Supplementary Information. Plasmids are available through Addgene (https://www.addgene.org/David_Savage/).

Code availability

The code for the experiments in this paper is available through GitHub (<https://github.com/junhaobearxiong/proteinguide/>). We have also published the ProteinGen python package (<https://ishangaur.com/proteingen/>) to facilitate the use of a variety of library design workflows, including ProteinGuide with popular generative and predictive models. We have also published a guide to using ProteinGuide (<https://ishangaur.com/proteinguide/>), which will be updated periodically with our latest practical tips for successful library design.

References

72. Ho, J., Jain, A. & Abbeel, P. Denoising diffusion probabilistic models. In *Proc. 33rd Conference on Advances in Neural Information Processing Systems* (eds Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F. & Lin, H.) (ACM, 2020).
73. Gillespie, D. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.* **81**, 2340–2361 (1977).
74. Gillespie, D. Approximate accelerated stochastic simulation of chemically reacting systems. *J. Phys. Chem.* **115**, 1716–1733 (2001).
75. Hoogeboom, E., Nielsen, D., Jaini, P., Forré, P. & Welling, M. Argmax flows and multinomial diffusion: learning categorical distributions. In *Proc. 34th Conference on Advances in Neural Information Processing Systems* (eds Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P. S. & Wortman Vaughan, J.) (ACM, 2021).
76. Albergo, M. & Eijnden, E. Building normalizing flows with stochastic interpolants. In *Proc. 11th International Conference on Learning Representations* (ed. Liu, Y.) (ICLR, 2023).
77. Lipman, Y., Chen, R., Hamu, H., Nickel, M. & Le, M. Flow matching for generative modeling. In *Proc. 11th International Conference on Learning Representations* (ed. Liu, Y.) (ICLR, 2023).
78. Liu, X., Gong, C. & Liu, Q. Flow straight and fast: learning to generate and transfer data with rectified flow. In *Proc. 11th International Conference on Learning Representations* (ed. Liu, Y.) (ICLR, 2023).
79. Kingma, D. & Gao, R. Understanding diffusion objectives as the ELBO with simple data augmentation. In *Proc. 36th Conference on Advances in Neural Information Processing Systems* (eds Oh, A. et al.) (ACM, 2023).
80. Peng, F. et al. Path planning for masked diffusion model sampling. Preprint at <https://doi.org/10.48550/arXiv.2502.03540> (2025).
81. Devlin, J., Chang, M., Lee, K. & Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proc. 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (eds Burstein, J., Doran, C. & Solorio, T.) (ACL, 2019).
82. Strauss, R. & Oliva, J. Arbitrary conditional distributions with energy. In *Proc. 34th Conference on Advances in Neural Information Processing Systems* (eds Ranzato, M. et al.) (ACM, 2021).
83. Karras, T. et al. Guiding a diffusion model with a bad version of itself. In *Proc. 37th Conference on Advances in Neural Information Processing Systems* (eds Zhang, C. et al.) (ACM, 2024).
84. Grathwohl, W., Swersky, K., Hashemi, M., Duvenaud, D. & Maddison, C. Oops I took a gradient: scalable sampling for discrete distributions. In *Proc. 38th International Conference on Machine Learning* (eds Meila, M. & Zhang, T.) (PMLR, 2021).
85. Schiff, Y. et al. Simple Guidance Mechanisms for Discrete Diffusion Models. In *Proc. 13th International Conference on Learning Representations* (ed. Yue, Y.) (ICLR, 2025).
86. Chen, Z. et al. Fast sampling via discrete non-Markov diffusion models with predetermined transition time. In *Proc. 37th Conference on Advances in Neural Information Processing Systems* (eds Zhang, C. et al.) (ACM, 2024).

87. Zheng, K. et al. Masked diffusion models are secretly time-agnostic masked models and exploit inaccurate categorical sampling. In *Proc. 13th International Conference on Learning Representations* (ed. Yue, Y.) (ICLR, 2025).
88. Del Moral, P. & Penev, S. *Stochastic Processes: From Applications to Theory* 1st edn (Chapman and Hall/CRC, 2017).
89. Stocco, F., Garibbo, M. & Ferruz, N. Steering generative models for protein design: aligning and conditioning strategies. *Curr. Opin. Struct. Biol.* **98**, 103250 (2026).
90. Dettmers, T., Pagnoni, A., Holtzman, A. & Zettlemoyer, L. QLoRA: efficient finetuning of quantized LLMs. In *Proc. 36th Conference on Advances in Neural Information Processing Systems* (eds Oh, A. et al.) (ACM, 2023).
91. Jiang, H. et al. SMART: robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. In *Proc. 58th Annual Meeting of the Association for Computational Linguistics* (eds Jurafsky, D. et al.) (ACL, 2020).
92. Dodge, J. et al. Fine-tuning pretrained language models: weight initializations, data orders, and early stopping. Preprint at <https://doi.org/10.48550/arXiv.2002.06305> (2020).
93. Mosbach, M., Andriushchenko, M. & Klakow, D. On the stability of fine-tuning BERT: misconceptions, explanations, and strong baselines. In *Proc. 9th International Conference on Learning Representations* (ed. Mohamed, S.) (ICLR, 2021).
94. Loshchilov, I., and Hutter, F. Decoupled Weight Decay Regularization. In *Proc. 7th International Conference on Learning Representations* (ed. Sainath, T.) (ICLR, 2019).
95. Hsu, C., Nisonoff, H., Fannjiang, C. & Listgarten, J. Learning protein fitness models from evolutionary and assay-labeled data. *Nat. Biotechnol.* **40**, 1114–1122 (2022).
96. Brookes, D., Park, H. & Listgarten, J. Conditioning by adaptive sampling for robust design. In *Proc. 36th International Conference on Machine Learning* (eds Chaudhuri, K. & Salakhutdinov, R.) (PMLR, 2019).
97. UniProt Consortium UniProt: the universal protein knowledgebase in 2021. *Nucleic Acids Res.* **49**, D480–D489 (2021).
98. Sanderson, T., Bileschi, M., Belanger, D. & Colwell, L. ProtelInfer, deep neural networks for protein functional inference. *eLife* **12**, e80942 (2023).
99. Yu, T. et al. Enzyme function prediction using contrastive learning. *Science* **379**, 1358–1363 (2023).
100. Dawson, N. et al. CATH: an expanded resource to predict protein function through structure and sequence. *Nucleic Acids Res.* **45**, D289–D295 (2017).
101. Geffner, T. et al. Proteina: scaling flow-based protein structure generative models. In *Proc. 13th International Conference on Learning Representations* (ed. Yue, Y.) (ICLR, 2025).
102. Kingma, D. & Ba, J. Adam: a method for stochastic optimization. In *Proc. 3rd International Conference on Learning Representations* (eds Bengio, Y. & LeCun, Y.) (ICLR, 2015).
103. Grandini, M., Bagli, E. & Visani, G. Metrics for multi-class classification: an overview. Preprint at <https://doi.org/10.48550/arXiv.2008.05756> (2020).
104. Yim, J. et al. SE(3) diffusion model with application to protein backbone generation. In *Proc. 40th International Conference on Machine Learning* (eds Krause, A. et al.) (PMLR, 2023).
105. Budzko, L., Sobiech, K., Jackowiak, P. & Figlerowicz, M. Engineered deaminases as a key component of DNA and RNA editing tools. *Mol. Ther. Nucleic Acids* **34**, 102062 (2023).
106. Del Arco, J., Acosta, J. & Lucas, J. Biotechnological applications of purine and pyrimidine deaminases. *Biotechnol. Adv.* **77**, 108473 (2024).
107. Hopf, T. et al. Mutation effects predicted from sequence co-variation. *Nat. Biotechnol.* **35**, 128–135 (2017).
108. Paszke, A. et al. Pytorch: an imperative style, high-performance deep learning library. In *Proc. 32nd Conference on Advances in Neural Information Processing Systems* (eds Wallach, H.) (ACM, 2019).
109. Ansel, J. et al. PyTorch 2: faster machine learning through dynamic Python bytecode transformation and graph compilation. In *Proc. 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (eds Abu-Ghazaleh, N., Gupta, R., Musuvathi, M. & Tsafir, D.) (ACM, 2024).
110. Xiong, J., Nisonoff, H. & Gaur, I. ProteinGuide data. *Zenodo* <https://doi.org/10.5281/zenodo.15635060> (2026).

Acknowledgements

We thank the Innovative Genomics Institute NGS facility and N. Krishnappa for assistance with Illumina sequencing. We thank the University of California, Berkeley DNA Sequencing facility for assistance with nanopore and Sanger sequencing. We thank H. Jiang, N. Ayala, A. Koehl, J. C. Li and F. Z. Wang for helpful discussions.

Author contributions

Conceptualized the study, J.X., M.L., H.N., D.F.S. and J.L. Designed and conducted the experiments, J.X., I.G., M.L., H.N. and L.M.O. Developed the algorithm, J.X., I.G. and H.N. Wrote the initial draft of the manuscript, J.X., I.G., M.L., H.N. and J.L. Analyzed the data, J.X., I.G., M.L. and H.N. Approved the final manuscript, all authors.

Funding

This work was supported in part by the US Department of Energy, Office of Science, Office of Biological and Environmental Research, Lawrence Livermore National Laboratory BioSecure Science Focus Area within the Secure Biosystems Design program (SCW1710), the Office of Naval Research under grant N00014-23-1-2587, the National Institutes of Health (NIH) National Institute of Mental Health under grant UF1MH130700, the NIH National Institute of Neurological Disorders and Stroke under grant R01NS126397 and the Chan Zuckerberg Initiative. D.F.S. is an investigator of the Howard Hughes Medical Institute. M.L. is a Life Science Research Fellow funded by the Howard Hughes Medical Institute.

Competing interests

D.F.S. is a cofounder and scientific advisory board member of Scribe Therapeutics. H.N. is currently an employee of Generate:Biomedicines. J.L. was a scientific advisory board member of Fable Therapeutics. The other authors declare no competing interests.

Additional information

Extended data is available for this paper at <https://doi.org/10.1038/s41587-026-03207-z>.

Supplementary information The online version contains supplementary material available at <https://doi.org/10.1038/s41587-026-03207-z>.

Correspondence and requests for materials should be addressed to David F. Savage or Jennifer Listgarten.

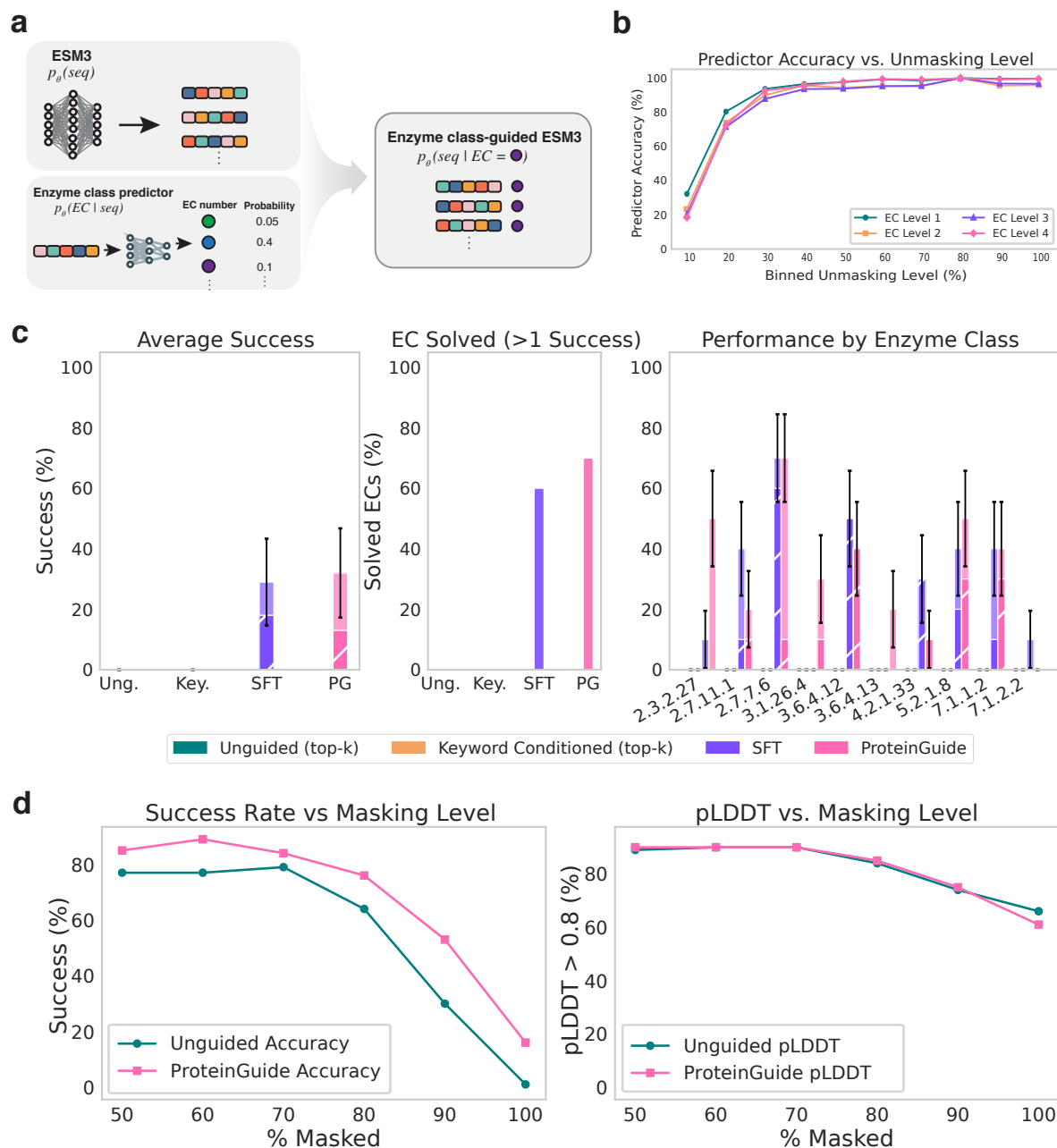
Peer review information *Nature Biotechnology* thanks the anonymous reviewers for their contribution to the peer review of this work. Peer reviewer reports are available.

Reprints and permissions information is available at www.nature.com/reprints.

Extended Data Table 1 | Summary of design tasks in this work (in silico and in vivo)

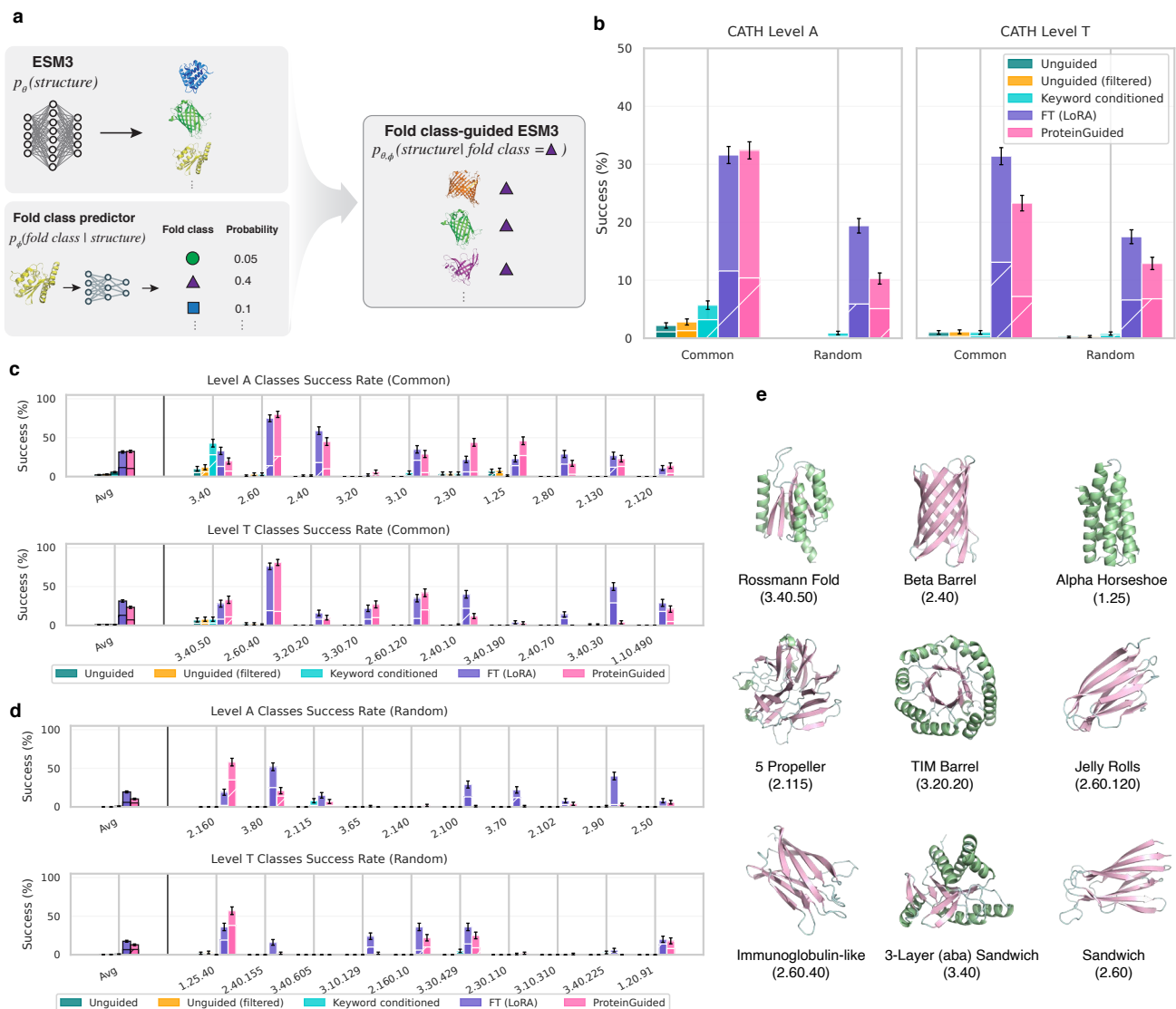
Task	Property	Design Setting	Multi-family?	# variants (predictor)	Mutational distance in dataset	Pre-trained model(s)
Stability-guided inverse folding	Folding stability ↑ ⁵³	Interpolative	Yes	~ 669k total, ~ 1-7k per protein	≤2 (per protein)	Protein MPNN ¹
CreiLOV assay-informed design	Fluorescence intensity ↑ ⁵⁴	Extrapolative	No	~ 167k (2k)	≤15	ESM3 ⁴⁷
UBE4B assay-informed design	Ubiquitination activity ↑ ⁵⁵	Extrapolative	No	~ 32k (2k)	≤6	ESM3
PABP/Pab1 assay-informed design	Functional complementation ↑ ⁵⁶	Extrapolative	No	~ 38k (2k)	≤2	ESM3
PbrR Pareto extrapolative design	Pb binding ↑; Zn binding ↓ ⁵⁷	Multi-property Pareto extrapolative	No	1060	≤5	ESM C ¹⁶
Enzyme-class guidance	EC label (class) ⁹⁷	Interpolative	Yes	~ 316k sequences, ~ 10 ² to ~ 10 ³ sequences for common ECs, ~ 10 ⁵ non-enzymes	N/A	ESM3
Fold-class guidance	CATH label (class) ¹⁰⁰	Interpolative	Yes	~ 200k structures total, ~ 10 ² to ~ 10 ⁴ per class	N/A	ESM3 (backbone structure)
In vivo: ABE design	Base editing activity ↑	Extrapolative	No	~ 2k	≤50	FMIF ¹⁷ , ESM3

The arrows in the 'Property' column indicate the desired direction we wish to guide the generation towards. For folding stability, we desire enhanced stability, which is represented by lower $\Delta\Delta G$. The 'Multi-family' column indicates whether multiple protein families are included in the dataset considered. Parentheses in the column '# variants (predictor)' indicate the number of labeled variants used to train the guidance predictor when it differs from the full dataset size. 'Mutational distance in dataset' represents the range of Hamming distances of the variants in a dataset relative to the wild-type sequence, when applicable. The 'Pre-trained model(s)' column indicate the pre-trained model(s) used with ProteinGuide.



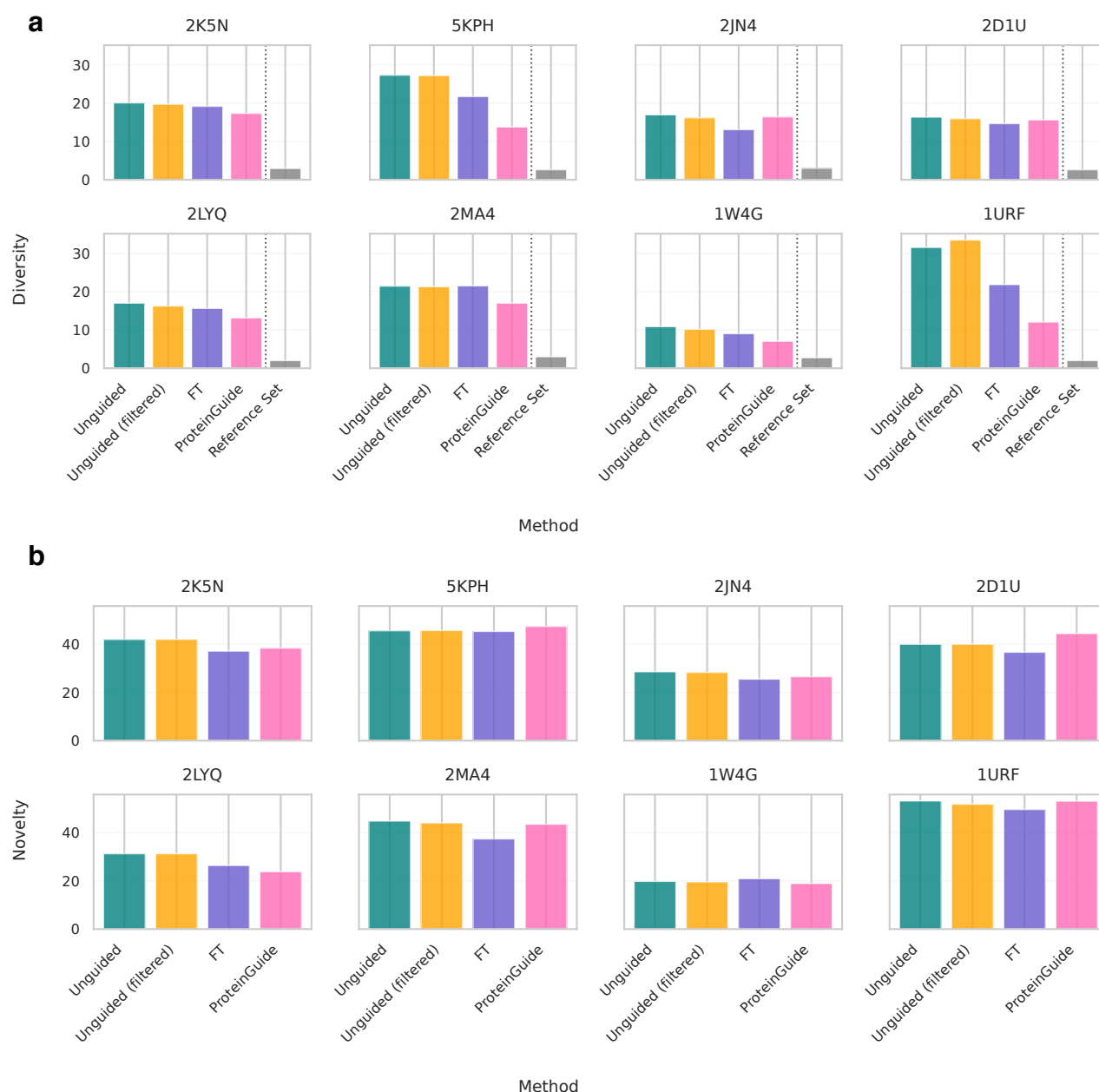
Extended Data Fig. 1 | Guiding ESM3 with enzyme annotations. **a**) Schematic for using ProteinGuide to guide ESM3 with an enzyme class predictor. **b**) Classification accuracy of the predictor on SwissProt test sequences as an increasing number of positions are unmasked. All accuracies are class balanced. **c**) Four generative methods (in legend) are assessed across 10 of the most common enzyme classes in SwissProt (horizontal axis of rightmost sub-figure). Left panel shows average success rate across all 10 common classes. Center shows the number of ECs for which each method produced at least one success. Right panel shows success rate by enzyme class for the 10 most common ECs. Top-k experiments generated 210 samples total. Out of these, the top 10 sequences

rated by the predictor as most likely to be in the target EC are kept for evaluation. The height of each bar represents the percentage of sequences predicted by the oracle to be of the target EC. The hatched sub-portion are the fraction of sequences in the target EC that are also predicted to fold with pLDDT ≥ 0.8 . Error bars are standard errors for proportions. **d**) Guidance performance when redesigning test set enzyme sequences. The x-axis of each subplot is the percent of positions in the original enzyme that were masked. The left subplot y-axis shows the percent of sequences generated (10 total) that were reclassified to be in the target enzyme class. The right subplot y-axis shows the percent of these 10 sequences that had a pLDDT greater than 0.8.



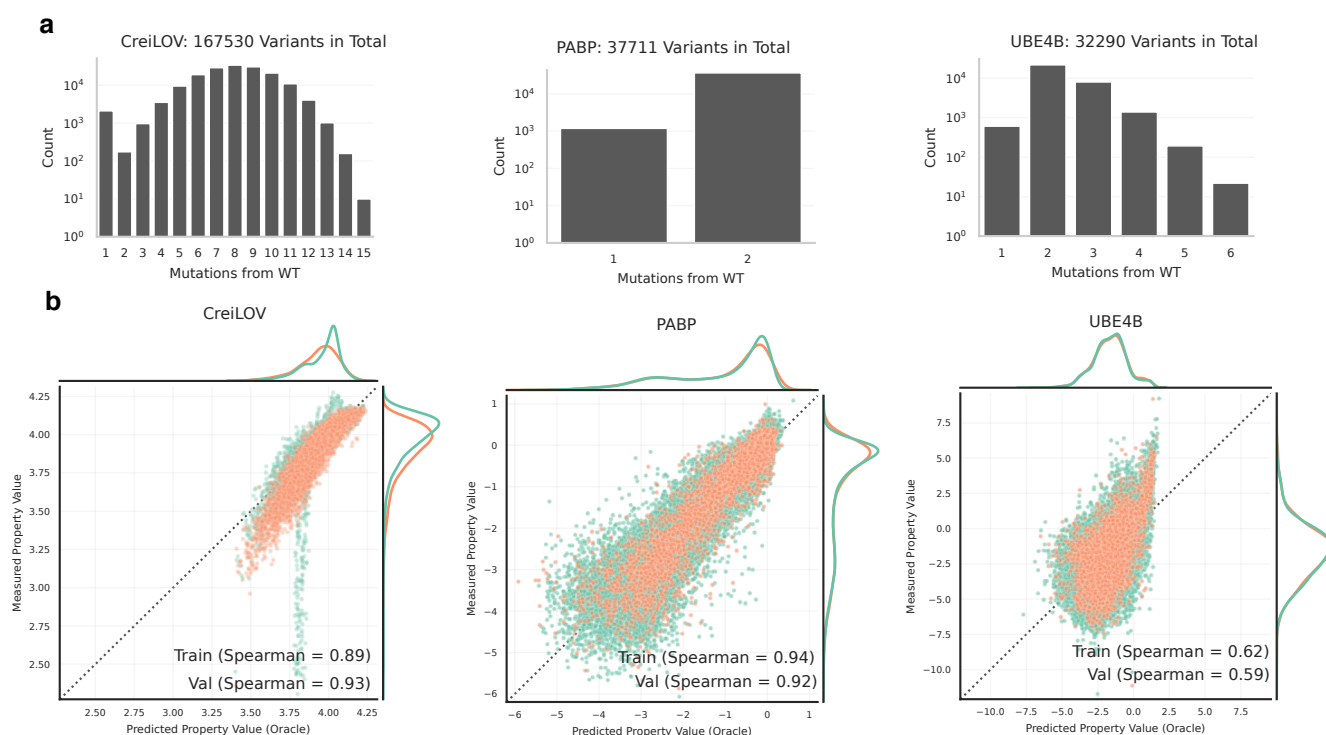
Extended Data Fig. 2 | Guiding ESM3 to generate backbone structures with CATH fold class annotations. a Schematic for using ProteinGuide to guide ESM3 with a fold class predictor (trained on CATH) to generate structures predicted to belong to specific fold class. **b** Aggregated success rates of different methods (in legends) across different CATH hierarchy levels. The height of each bar shows the percentage of structures correctly classified to their target fold class, while the hatched sub-portion indicates the subset that are also designable (self-consistency RMSD ≤ 2 Å). Results are grouped by CATH level, and the classes for each level were grouped by their frequency of occurrence in the training dataset of the fold class oracle. For each target fold class, methods are evaluated

on $k = 100$ generated samples. Metrics are aggregated within each group following Geffner et al.¹⁰¹. 'Common' consists of the 10 most common classes for each level, and 'Random' consists of 10 randomly selected classes for each level. **c** Per-class success rates of different methods for the Common classes for both A level (top) and T level (bottom). **d** Per-class success rates of different methods for the Random classes for both A level (top) and T level (bottom). **e** Examples of generated backbone structures by guiding ESM3 with some example CATH classes. The shown examples are both classified correctly to the target fold class by the fold classifier and are designable.

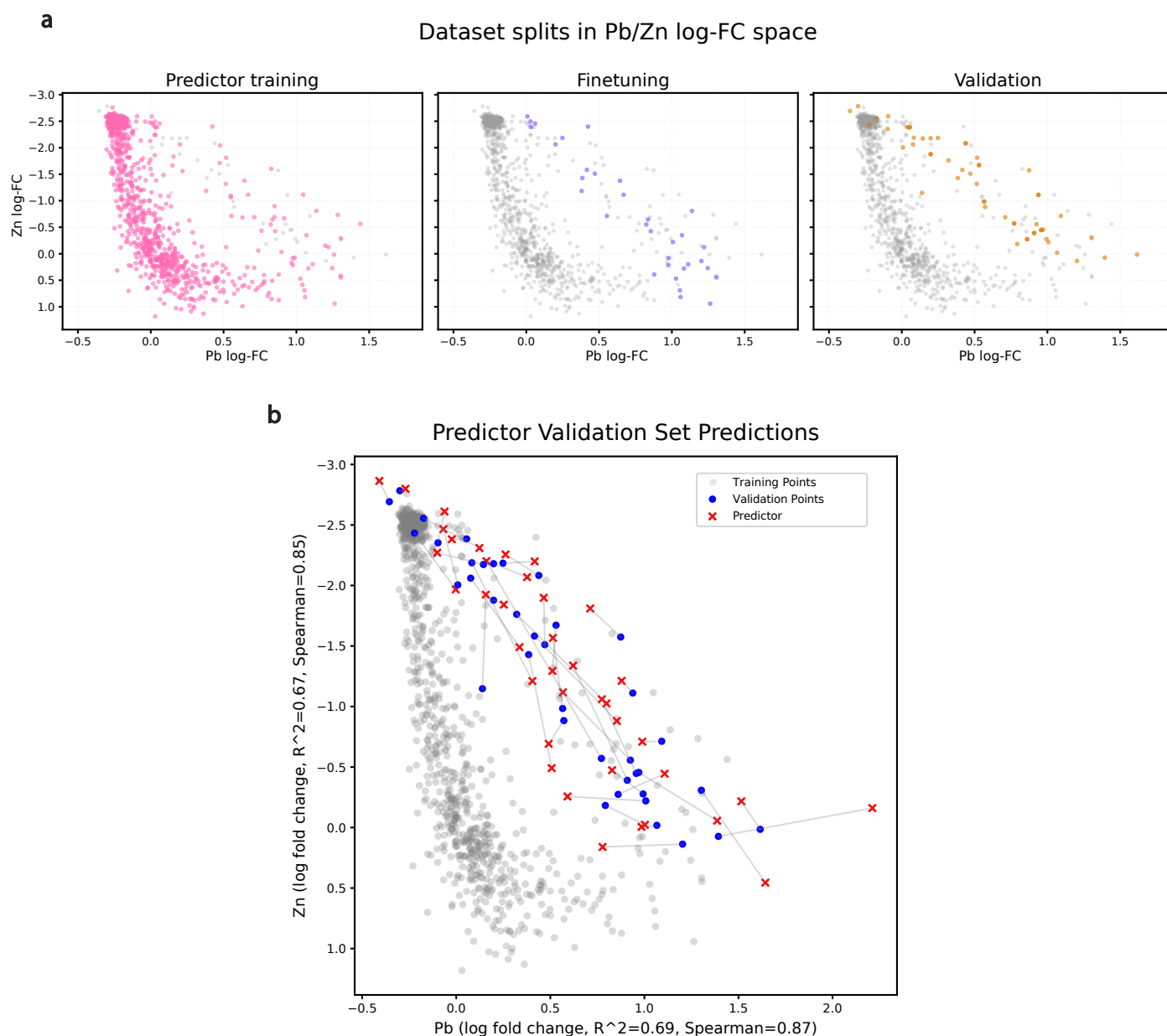


Extended Data Fig. 3 | Diversity and novelty of generated sequences for all methods in the ProteinMPNN stability experiment. a) Diversity of the generated sequences. For each method, diversity is evaluated on $k = 100$ sequences. Diversity is computed as the averaged pairwise Hamming distance among the generated sequences. For each protein, the 'Reference Set' diversity

is computed on 500 randomly sampled sequences from the labeled dataset for each protein. **b)** Novelty of the generated sequences. For each method, novelty is evaluated on $k = 100$ sequences. For each sequence, novelty is computed as its minimum Hamming distance with sequences in the full labeled dataset. For each method, the mean novelty among the generated sequences is shown.

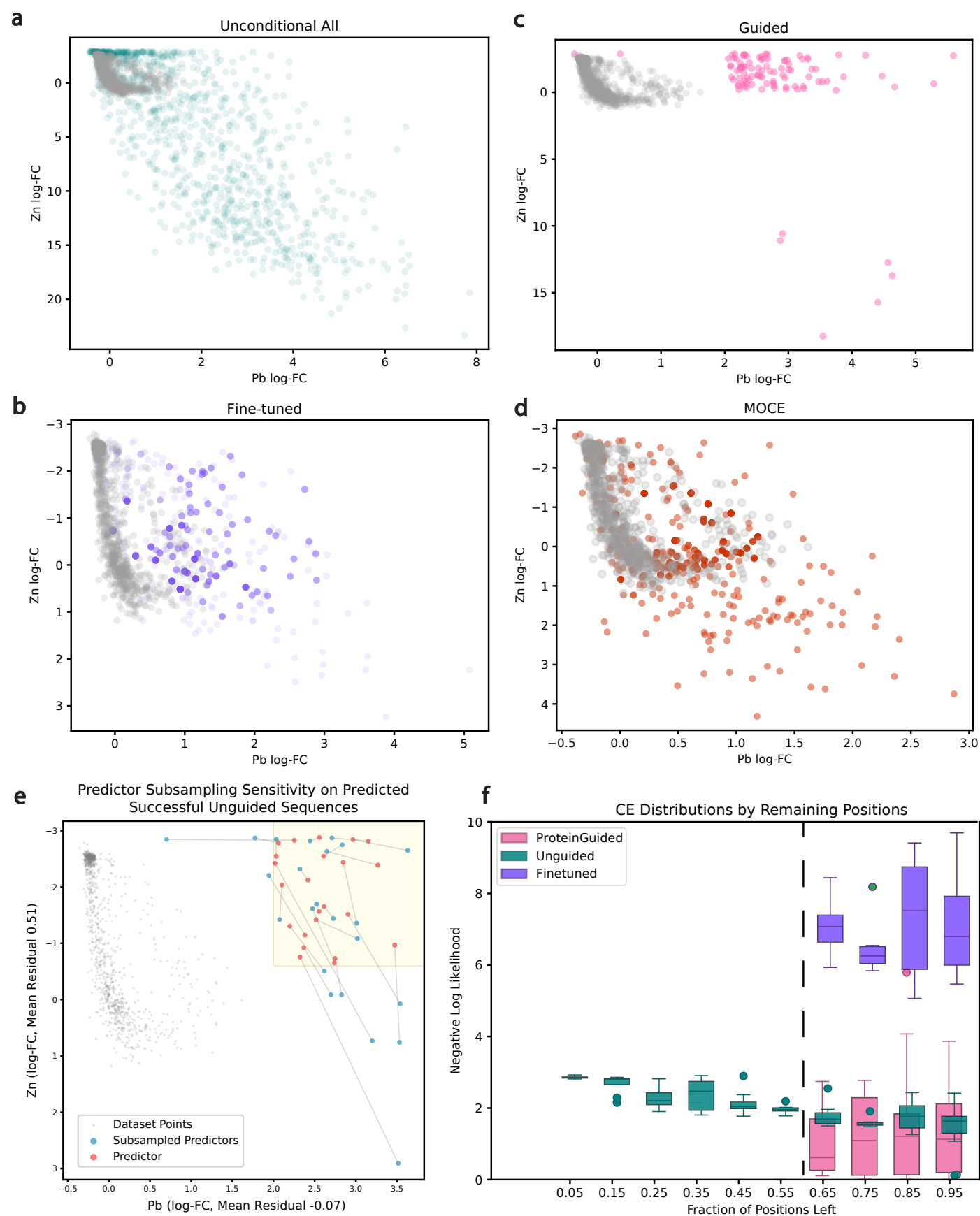


Extended Data Fig. 4 | Datasets and oracle models used for the extrapolative experiment of guiding ESM3 with assay-informed predictive models. a) Distribution of variant counts in each dataset stratified by the Hamming distance from the wild-type sequence. **b)** Scatterplot of the labeled vs. oracle predicted property value in each dataset, for both the training and validation set.



Extended Data Fig. 5 | Data splits used in the multi-property, Pareto-extrapolative PbrR experiment. **a)** Summary of training and validation splits used for the PbrR experiment. Gray points show full dataset points not included in that split. The validation split was common between the predictor

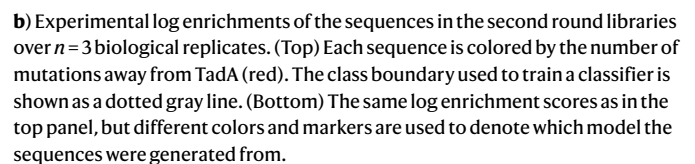
and fine-tuned model. The fine-tuning training data was the portion of the predictor training data that was above the dashed on-/off-target threshold line shown in the center subpanel. **b)** Predictions from the regression model used for ProteinGuide compared with the true assay value on the validation set.



Extended Data Fig. 6 | See next page for caption.

Extended Data Fig. 6 | Multi-property, Pareto-extrapolative experiment: unfiltered generation results, heuristic constraint adjustment, and negative log-likelihood curves. a–d Show the unfiltered compute-matched sequences generated by the three methods benchmarked in the main text and MOCE, the method used by Wang et al.⁵⁷. **e** Example of training-data-subsampled predictors (blue) against the predictor used for guidance (red). The residuals are ‘Predictor’ minus ‘Subsampled’. Residuals were calculated on randomly sampled sequences

of up to 30 mutations that were predicted to be in the target region for the Pareto-extrapolative task (Fig. 4b, yellow box). **f** Time-series of negative log likelihoods for each method on the set of sequences predicted by the oracle to be in the target region of the Pareto-extrapolative task (Fig. 4b, yellow box). Dashed line shows the time point from which generation started. A cross entropy of 3 on the y-axis corresponds to the negative log-likelihood on the data for a uniform distribution over amino acids.



Reporting Summary

Nature Portfolio wishes to improve the reproducibility of the work that we publish. This form provides structure for consistency and transparency in reporting. For further information on Nature Portfolio policies, see our [Editorial Policies](#) and the [Editorial Policy Checklist](#).

Statistics

For all statistical analyses, confirm that the following items are present in the figure legend, table legend, main text, or Methods section.

n/a	Confirmed
☐	☒ The exact sample size (<i>n</i>) for each experimental group/condition, given as a discrete number and unit of measurement
☐	☒ A statement on whether measurements were taken from distinct samples or whether the same sample was measured repeatedly
☒	☐ The statistical test(s) used AND whether they are one- or two-sided <i>Only common tests should be described solely by name; describe more complex techniques in the Methods section.</i>
☒	☐ A description of all covariates tested
☒	☐ A description of any assumptions or corrections, such as tests of normality and adjustment for multiple comparisons
☐	☒ A full description of the statistical parameters including central tendency (e.g. means) or other basic estimates (e.g. regression coefficient) AND variation (e.g. standard deviation) or associated estimates of uncertainty (e.g. confidence intervals)
☒	☐ For null hypothesis testing, the test statistic (e.g. <i>F</i> , <i>t</i> , <i>r</i>) with confidence intervals, effect sizes, degrees of freedom and <i>P</i> value noted <i>Give P values as exact values whenever suitable.</i>
☐	☒ For Bayesian analysis, information on the choice of priors and Markov chain Monte Carlo settings
☒	☐ For hierarchical and complex designs, identification of the appropriate level for tests and full reporting of outcomes
☐	☒ Estimates of effect sizes (e.g. Cohen's <i>d</i> , Pearson's <i>r</i>), indicating how they were calculated

Our web collection on [statistics for biologists](#) contains articles on many of the points above.

Software and code

Policy information about [availability of computer code](#)

Data collection	Illumina sequencing was performed at the Innovative Genomics Institute NGS facility. Nanopore and Sanger sequencing were performed at the UC Berkeley DNA Sequencing Facility.
Data analysis	The code for the experiments in this paper is available at https://github.com/junhaobearxiong/proteinguide/ . We have also published the ProteinGen python package at https://ishangaur.com/proteingen/ to facilitate the use of a variety of library design workflows, including, but not limited to, ProteinGuide with popular generative and predictive models. We have also published a guide to using ProteinGuide at https://ishangaur.com/proteinguide/ , which will be updated periodically with our latest practical tips for successful library design.

For manuscripts utilizing custom algorithms or software that are central to the research but not yet described in published literature, software must be made available to editors and reviewers. We strongly encourage code deposition in a community repository (e.g. GitHub). See the Nature Portfolio [guidelines for submitting code & software](#) for further information.

Data

Policy information about [availability of data](#)

All manuscripts must include a [data availability statement](#). This statement should provide the following information, where applicable:

- Accession codes, unique identifiers, or web links for publicly available datasets
- A description of any restrictions on data availability
- For clinical datasets or third party data, please ensure that the statement adheres to our [policy](#)

Source data associated with the code are available on Zenodo (<https://zenodo.org/records/15635060>). Raw sequencing data for the base editor experiment are available under BioProject PRJNA1467757 (<https://dataview.ncbi.nlm.nih.gov/object/PRJNA1467757>). Primers and protein sequences are available in the Supplementary Information. Plasmids are available at Addgene (https://www.addgene.org/David_Savage/)

Research involving human participants, their data, or biological material

Policy information about studies with [human participants or human data](#). See also policy information about [sex, gender \(identity/presentation\), and sexual orientation](#) and [race, ethnicity and racism](#).

Reporting on sex and gender	N/A
Reporting on race, ethnicity, or other socially relevant groupings	N/A
Population characteristics	N/A
Recruitment	N/A
Ethics oversight	N/A

Note that full information on the approval of the study protocol must also be provided in the manuscript.

Field-specific reporting

Please select the one below that is the best fit for your research. If you are not sure, read the appropriate sections before making your selection.

☒ Life sciences ☐ Behavioural & social sciences ☐ Ecological, evolutionary & environmental sciences

For a reference copy of the document with all sections, see nature.com/documents/nr-reporting-summary-flat.pdf

Life sciences study design

All studies must disclose on these points even when the disclosure is negative.

Sample size	No statistical methods were used to predetermine sample size. For the in vivo base editor experiments, library experiments were performed in duplicates or triplicates and individual variants testing were performed in triplicates. For in silico experiments, sample sizes were determined by the available published datasets and generation budgets are detailed in the Methods.
Data exclusions	No data were excluded from the analyses. For the in vivo base editor experiment, all sequenced variants were included in the enrichment analysis. For in silico experiments, no data were excluded from evaluation.
Replication	The in vivo base editor experiment was performed with two biological replicates for the first-round library and three biological replicates for the second-round library. All attempts at replication were successful and results were consistent across replicates. Data from all replicates is deposited in the NCBI public depository.
Randomization	Randomization is not directly relevant to this study. Library variants were designed computationally and screened in a pooled format. For in silico experiments, data were randomly split into training and evaluation sets as described in the Methods.
Blinding	Blinding is not relevant to this study. Wet-lab activity measurements were obtained via high-throughput sequencing-based readout without subjective assessment. Computational evaluations used automated pipelines.

Reporting for specific materials, systems and methods

We require information from authors about some types of materials, experimental systems and methods used in many studies. Here, indicate whether each material, system or method listed is relevant to your study. If you are not sure if a list item applies to your research, read the appropriate section before selecting a response.

Materials & experimental systems

n/a	Involved in the study
☒	☐ Antibodies
☒	☐ Eukaryotic cell lines
☒	☐ Palaeontology and archaeology
☒	☐ Animals and other organisms
☒	☐ Clinical data
☒	☐ Dual use research of concern
☒	☐ Plants

Methods

n/a	Involved in the study
☒	☐ ChIP-seq
☒	☐ Flow cytometry
☒	☐ MRI-based neuroimaging

Plants

Seed stocks

Report on the source of all seed stocks or other plant material used. If applicable, state the seed stock centre and catalogue number. If plant specimens were collected from the field, describe the collection location, date and sampling procedures.

Novel plant genotypes

Describe the methods by which all novel plant genotypes were produced. This includes those generated by transgenic approaches, gene editing, chemical/radiation-based mutagenesis and hybridization. For transgenic lines, describe the transformation method, the number of independent lines analyzed and the generation upon which experiments were performed. For gene-edited lines, describe the editor used, the endogenous sequence targeted for editing, the targeting guide RNA sequence (if applicable) and how the editor was applied.

Authentication

Describe any authentication procedures for each seed stock used or novel genotype generated. Describe any experiments used to assess the effect of a mutation and, where applicable, how potential secondary effects (e.g. second site T-DNA insertions, mosaicism, off-target gene editing) were examined.