Protein Design via Discrete Walk-Jump Sampling

Anisha Iyer*
Department of Electrical Engineering and Computer Science
UC Berkeley
Berkeley, CA 94720
`anishaiyer@berkeley.edu`

Abstract

Protein design offers immense potential to address critical challenges in healthcare, climate change, and other issues of international societal impact. Recent advancements in generative modeling enables an avenue for innovation with protein design framed as a problem of discrete sequence generation. Discrete Walk-Jump Sampling (dWJS) has demonstrated success in overcoming common bottlenecks in training and sampling to design diverse and high-quality antibodies, while outperforming several baseline methods such as masked language models, diffusion models, and autoregressive language models. This project aims to recreate the findings in the original dWJS paper and experiments with adjustments to hyperparameters to increase efficiency, decrease burden of computationally expensive attention heads, and assess model performance. I also experimented with learning rate, batch size, number of workers, learning rate scheduler, and noise hyperparameter combinations to determine optimal configurations, conducting several hyperparameter sweeps for a denoiser model and an energy-based model. Additionally, I made changes to specific coding implementations and added scripts to perform statistical analyses on generated samples This is especially important to provide a foundation for future use of dWJS on protein engineering problems beyond antibodies to unlock the broader potential of dWJS for protein design.

1 Introduction

Endogenous protein engineering plays a fundamental role in nearly all biological processes, with significant promise to aid in unsolved problems in medicine, bioengineering, and climate change [Alley et al., 2019]. With recent advances in generative modeling, protein design has evolved rapidly through the strengths of deep learning and probabilistic models [Jumper, 2021]. As protein design can be formulated as a discrete sequence generation problem, the authors seek to infer useful proteins in a large, discrete, and sparsely functional protein space, which is exponentially increasing in dimensionality with sequence length [Ingraham, 2019]. Recently, innovation in machine learning-guided protein design has been fueled by advancements in discrete generative models such as masked language models, autoregressive models, and diffusion-based approaches [Rives, 2021]. Masked language models, and autoregressive models, effectively model sequence context, while diffusion models capture stochasticity in sequence generation [Dhariwal and Nichol, 2021]. However, autoregressive protein design methods are inefficient, have high inference latency, and suffer from error accumulation [Madani, 2023]. Conversely, current non-autoregressive diffusion models are are also ineffecient and are not well-optimized for real protein discovery and design tasks [Hoogeboom, 2021].

For the specific problem of protein design, generative models must produce novel, unique, and diverse outputs within the constraints of the problem space, from an accurate representation of the underlying

*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

data distribution. Energy-based models (EBMs) capture the data distribution fitting an energy function that formulates a probability distribution analogously to the Boltzmann distribution in statistical physics [LeCun, 2006]. EBMs can be difficult to train and sample from, which is overcome by recent advancements in diffusion models and similar formulations of denoising objectives based on score matching [Song, 2021]. However, these models encounter limitations with efficient sampling and accurate modeling of combinatorial spaces when applied to discrete sequence generation

Seeking to provide an efficient, non-autoregressive, discrete generative modeling framework for sampling from discrete generative models, the authors developed Smoothed Discrete Sampling (SDS). Building on the neural empirical Bayes (NEB) framework, the authors proposed discrete Walk-Jump Sampling (dWJS) in concert with SDS, to overcome the brittleness of discrete EBMs and diffusion models and provide a robust, general framework for protein design [Dieleman, 2023]. Furthermore, they introduced the Distributional Conformity Score (DCS), a novel metric to evaluate the quality of protein samples. Their approach demonstrated the viability of EBMs for discrete distribution modeling and suggested that diffusion models with multiple noise scales may not be necessary for effective protein discovery [Song, 2021].

Discrete Walk-Jump Sampling (dWJS) demonstrates state-of-the-art performance in antibody design, achieving a better balance between diversity and functionality. This study replicates dWJS's original findings and attempts to reduce computationally expensive training requirements further with the ultimate goal of supporting generalizability of dWJS to other protein families, such as enzymes. In this project, I replicate the findings of the authors of dWJS. Specifically, I focus on performing hyperparameter sweeps to find the optimal noise parameter for EBMs, implementing evaluation scripts to assess the quality of generated protein samples, exploring trade offs between learning rate and training time, and verifying the robustness of the methods for discrete sequence generation.

2 Background

2.1 Energy-Based Models

Energy-based models (EBMs) leverage the Boltzmann distribution from statistical physics to learn an energy function $f_\theta : X \to \mathbb{R}$ mapping inputs x (in $\mathbb{R}^d$) to a scalar "energy" value [LeCun, 2006]. The data distribution is approximated by the Boltzmann distribution:

$$p_\theta(x) \propto e^{-f_\theta(x)}.$$

EBMs are usually trained with contrastive divergence, and new samples are drawn from the distribution $p_\theta(x)$ by Markov-Chain Monte Carlo (MCMC). Details of the loss function used in this work are given in Section 3. In Langevin MCMC, samples are initialized from a known data point or random noise and refined with discretized Langevin diffusion [**?**]:

$$x_{k+1} = x_k - \delta \nabla f_\theta(x_k) + \sqrt{2\delta}\epsilon_k, \quad \epsilon_k \sim \mathcal{N}(0, I_d),$$

where ∇ denotes the gradient of the energy function with respect to inputs, k is the sampling step, δ is the discretization step size, and the noise ϵ_k is drawn from the normal distribution at each step.

2.2 Neural Empirical Bayes

In Neural Empirical Bayes (NEB), the random variable X is transformed with additive Gaussian noise:

$$Y = X + N(0, \sigma^2 I_d).$$

The least-squares estimator of X given $Y = y$ is:

$$\hat{x}(y) = y + \frac{\sigma^2}{2} \nabla \log p(y),$$

where $p(y) = \int p(y|x)p(x)\, dx$ is the probability distribution function of the smoothed density. This estimator is often expressed directly in terms of $g(y) = \nabla \log p(y)$, known as the score function **?**, which is parameterized with a neural network denoted by $g_\phi : \mathbb{R}^d \to \mathbb{R}^d$. The least-squares estimator then takes the following parametric form:

$$\hat{x}_\phi(y) = y + \frac{\sigma^2}{2} g_\phi(y).$$

Putting this all together leads to the following learning objective:

$$L(\phi) = \mathbb{E}_{x\sim p(x), y\sim p(y|x)} \left[\|x - \hat{x}_\phi(y)\|_2^2 \right],$$

which is optimized with stochastic gradient descent. Notably, no MCMC sampling is required during learning. In short, the objective is "learning to denoise" with an empirical Bayes formulation [Song, 2021].

3 Methods

3.1 Discrete Walk-Jump Sampling

Discrete Walk-Jump Sampling (dWJS) employs a two-stage mechanism for generating protein sequences. The *walk* phase introduces local perturbations to explore sequence neighborhoods in a smoothed energy manifold, while the *jump* phase serves to denoise the sequence that was more easily sampled back to a protein sequence like the input.

The EBM is trained with contrastive divergence on the manifold of smoothed, one-hot encodings, after adding noise. These one-hot encodings are given by $\mathbf{y}$, where $\mathbf{y} = \mathbf{x} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I}_d)$ where $\mathbf{x} \in \{0,1\}^d$ represents a one-hot encoded vector and $\boldsymbol{\epsilon}$ is Gaussian noise with variance σ^2. As a result, the EBM learns an energy function $f_\theta(\mathbf{y})$ to model the distribution of noisy encodings.

Separately, the denoising model is trained to recover $\mathbf{x}$ from $\mathbf{y}$ by optimizing the least-squares objective learning objective from Section 2.2. Inferred protein sequences, corresponding to new antibodies, are generated by sampling with Langevin MCMC to obtain the noisy sequences following gradients from the EBM, denoising with the least-squares estimator from the previous section, and recovering a one-hot encoding by taking the argmax of the denoised vector.

Sequence generation is performed using Langevin MCMC to sample noisy encodings $\mathbf{y}$ from the learned energy landscape: $\mathbf{y}_{k+1} = \mathbf{y}_k - \delta\nabla_{\mathbf{y}} f_\theta(\mathbf{y}_k) + \sqrt{2\delta}\boldsymbol{\epsilon}_k, \quad \boldsymbol{\epsilon}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d$

The sampled encodings are denoised using the least-squares estimator $\hat{\mathbf{x}}_\phi(\mathbf{y})$, and the final sequence is obtained by taking the $\arg\max$ over the denoised vector: $\hat{\mathbf{x}} = \arg\max_i \hat{\mathbf{x}}_\phi(\mathbf{y})_i$.

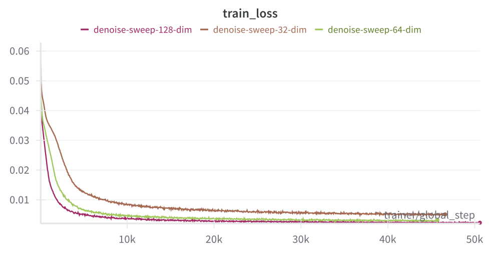
3.2 Datasets and Preprocessing

The dataset and preprocessing techniques are the same as in the original paper, in which the authors encode antibody protein molecules as $x = (x_1, ..., x_d)$, where $x_l \in \{1, ..., 20\}$ corresponds to the amino acid identity at position *l*. Preprocessing of the dataset from the Observed Antibody Space (OAS) database included aligning sequences according to the AHo numbering scheme using ANARCI and one-hot encoding. By aligning to address insertions and deletions, a gap token is added that can be introduced or removed to change sequence length during sampling. Overall, the datasets are tokenized and preprocessed according to the protocol in the original paper, to ensure compatibility with the dWJS framework.

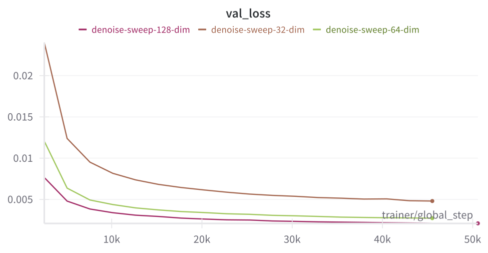
3.3 Experimental Setup and Model Configurations

All models were implemented and trained using PyTorch, based on the original dWJS implementations, with training pipelines streamlined via PyTorch Lightning, enabling efficient handling of model training, evaluation, and multi-GPU support. Experiment configurations were also managed using Hydra, which facilitated systematic hyperparameter tuning and reproducible workflows. For experiment tracking, logging, and visualization, Weights and Biases (WandB) was utilized, which provided seamless monitoring of metrics and performance across training runs and easy access to compare runs.

The authors originally adopted a uniform architecture for the Energy-Based Model (EBM), comprising three Conv1D layers with kernel sizes of 15, 5, and 3, and padding set to 1. Each Conv1D layer is followed by ReLU activations, and the final output is a linear layer with a size of 128. The denoising network utilizes a 35-layer ByteNet architecture with a hidden dimension of 128, trained from scratch. ByteNet has been shown to perform on par with transformers for protein sequence pretraining tasks.



(a) Training loss across different n_{dim} configurations.



(b) Validation loss across different n_{dim} configurations.

Figure 1: Loss convergence trends for training and validation. Lower-dimensional models exhibit comparable loss convergence to higher-dimensional ones.

Both models were trained using the AdamW optimizer implemented in PyTorch, with a batch size of 256 and an initial learning rate of 1×10^{-4} in the original paper. Training proceeded with early stopping to prevent overfitting.

3.4 Transformer Implementation of the Denoising Network

In addition to the ByteNet-based denoiser, the original dWJS contained an implementation of a transformer-based architecture with 12 hidden layers, 8 attention heads, feed-forward layers of dimensionality 2048, and encoder/decoder input features of size 256. The model employs SiLU activations throughout. The transformer-based implementation of the denoising network achieves performance comparable to ByteNet, underscoring the robustness of the method to architectural variations, provided the model is sufficiently expressive.

3.5 Model Combinations in Smoothed Discrete Sampling

Due to the decoupled nature of the walk and jump steps in Smoothed Discrete Sampling, there exist multiple viable implementations. Empirically, Algorithm 1 leverages both energy-based and score-based modeling to generate high-quality, novel, unique, and diverse samples. Below is a summary of four natural sampling strategies arising from different combinations of energy-based and score-based parameterizations:

1. **Discrete Walk-Jump Sampling (dWJS):** An EBM walk $f_\theta(\mathbf{y})$ is combined with a denoiser jump $g_\phi(\mathbf{y})$.
2. **Score-based dWJS:** The denoising network $g_\phi(\mathbf{y})$ is used for both walk and jump steps.
3. **Deep Energy Estimator Network (DEEN):** A denoiser is trained as the derivative of an energy model. This represents an energy-based formulation of a score-based generative model [**?**].
4. **dWJS-EBM:** Relies on an EBM for both sampling and denoising, with the gradient $\nabla f_\theta(\mathbf{y})$ used during denoising.

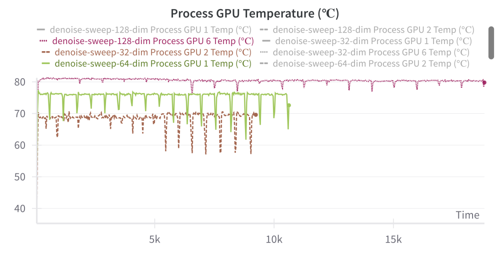
   All four combinations of these models are implemented in the original paper. I recreated energy-based and score-based dWJS as well as dWJS-EBM, but did not recreate DEEN due to time and resources constraints.

4 Results

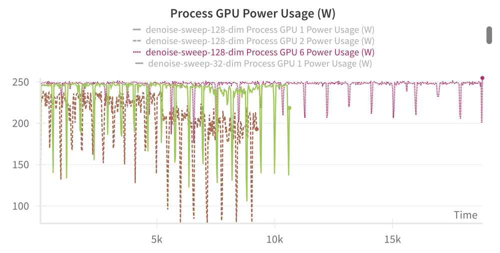
4.1 Computationally Expensive Denoiser Architecture

I evaluated the transformer-based denoising network across different n_{dim} values of 32, 64, and 128 to assess its performance in terms of convergence, computational efficiency, and energy consumption.

All configurations successfully converged to low loss values, demonstrating the robustness of the transformer architecture, even with only $n_{\text{dim}} = 32$. Notably, the $n_{\text{dim}} = 32$ model achieved loss convergence that was comparable to the higher-dimensional configurations ($n_{\text{dim}} = 64$ and



(a) GPU temperature across different n_{dim} configurations.



(b) GPU power consumption across different n_{dim} configurations.

Figure 2: Power and energy trends during training. Higher-dimensional models consume significantly more energy.

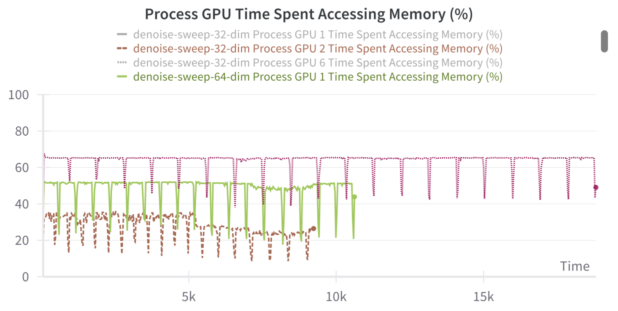


Figure 3: GPU access time across different n_{dim} configurations. Lower-dimensional models exhibit reduced computation time.

$n_{\text{dim}} = 128$). This suggests that lower-dimensional models can maintain denoising performance if the architecture is sufficiently expressive.

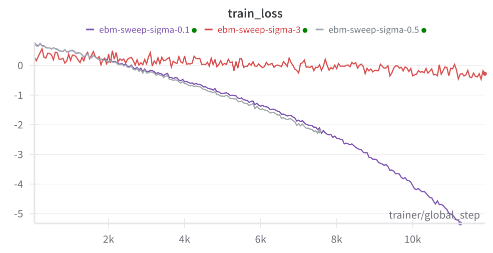
The higher-dimensional configurations exhibited significantly greater computational and energy requirements. Both $n_{\text{dim}} = 64$ and $n_{\text{dim}} = 128$ models showed sustained higher GPU power usage throughout training, as illustrated in Figure 2. Additionally, their training durations were longer compared to the $n_{\text{dim}} = 32$ model. In contrast, the $n_{\text{dim}} = 32$ model demonstrated substantially reduced GPU power consumption and computation time while maintaining competitive performance. These findings clearly demonstrate the trade-off between model dimensionality and computational efficiency.

4.2 EBM Hyperparameter Sweeps

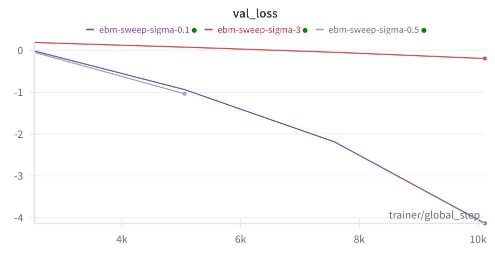
To optimize the Energy-Based Model (EBM), several hundred hyperparameter sweeps were conducted using WandB and manual grid search. A critical parameter was the starting the noise parameter, σ.

Through exensive trials with various σ values, Figure 4 demonstrates the training loss behavior across different σ values. It is evident that models with $\sigma \geq 0.5$ achieve smoother and more stable convergence, compared to the model with $\sigma = 0.1$, which exhibits slower convergence and higher loss values. Larger σ values appear to enable the model to better navigate the loss landscape, resulting in more effective training dynamics. Similarly, the validation loss trends reinforce the observation that models with larger σ values (0.5 and 3.0) achieve comparable performance, which is consistent with findings in the paper. This indicates that increasing σ promotes generalization and robustness, as the validation loss stabilizes at a lower level compared to $\sigma = 0.1$. This also demonstrates a benefit in only having one layer of noise as compared to diffusion models which have multiple levels of noise.

Through several hundred hyperparameter sweeps on WandDB and manual sweeps, it was determined that the best starting learning rate is 1×10^{-6} despite the increase in training time associated with the smaller step size. Starting the learning rate at 1×10^{-4} as in the original paper, or even 1×10^{-5},

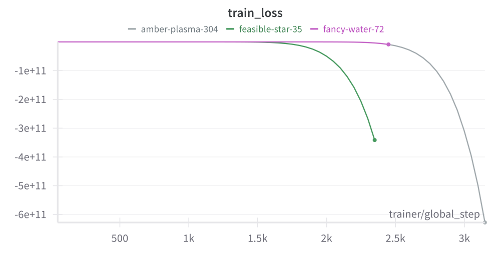


(a) Training loss across different σ values of 0.1, 0.5, 30.

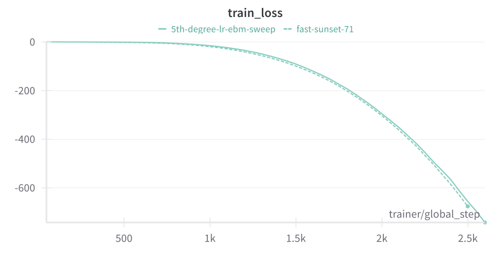


(b) Validation loss across different σ values of 0.1, 0.5, 30.

Figure 4: Loss convergence trends for training and validation. Models with sigma greater than 0.5 exhibit comparable loss convergence, as compared to those with sigma less than 0.5.



(a) Training loss with initial learning rate learning rate at 1×10^{-4}



(b) Training loss with initial learning rate learning rate at 1×10^{-5}.

Figure 5: Learning rate does not support loss convergence except with initial learning rate learning rate at 1×10^{-6}

consistently caused unstable divergence in the loss function, leading it to tend toward negative infinity. The divergence was particularly pronounced in configurations with lower n_{dim}, as the model's capacity to stabilize gradients was limited. This result underscores the importance of selecting a sufficiently small initial learning rate to ensure stability in loss convergence, particularly in transformer-based architectures designed for denoising tasks.

In addition to stabilizing convergence, the smaller learning rate allowed the model to better explore the optimization landscape, as evidenced by improved loss trajectories during training (see Figure 5). While this increased the training duration, it ultimately produced more robust results across all tested configurations. This likely prevented the model from overshooting the minimum or getting stuck in a valley in the loss landscape.

4.3 4.3 Wasserstein Distance and Protein Quality Metrics

I also evaluated several protein quality metrics against resultant sets of samples for each of the three implemented combinations of models for dWJS. I reimplemented some of the quantifications for this based on log probabilities from the original paper, and added a few scripts for statistical analysis, but only included a portion of these findings here as much of the data I obtained was not clearly comparable through visualizations or tables Table 1.

5 Discussion

Discrete Walk-Jump Sampling (dWJS) introduces a novel framework for protein sequence generation, effectively addressing the challenges associated with navigating the vast and sparsely functional protein sequence space. The walk-jump mechanism enables efficient exploration by balancing sequence diversity with functional constraints, achieving state-of-the-art performance in generating high-quality protein sequences. By systematically optimizing the noise parameter and conducting

Category	Value
dWJS (Heavy Chain)	3.267
dWJS (Light Chain)	3.382
Score-based dWJS (Heavy Chain)	1.783
Score-based dWJS (Light Chain)	1.937
dWJS-EBM (Heavy Chain)	3.267
dWJS-EBM (Light Chain)	3.382

Table 1: Average Wasserstein distance for samples generate from different SDS techniques.

comprehensive hyperparameter sweeps, we minimize computational costs and maintain robustness as dWJS positions itself as a scalable solution for protein design.

This framework represents a significant advancement over traditional approaches, such as masked language models and diffusion-based methods, which often struggle with difficulties with sampling, training, and modeling complex combinatorial spaces. Additionally, dWJS's ability to efficiently generate functional antibody sequences while maintaining computational feasibility underscores its potential as a transformative tool for generative protein design. Beyond antibodies, its adaptable architecture suggests broad applicability to other protein families, such as enzymes, which could pave the way for more impactful contributions to protein engineering and synthetic biology.

Despite the success of dWJS in antibody design, several challenges remain for future directions. One such challenge is optimizing dWJS for broader protein families, such as enzymes. Another future direction is incorporating biochemical constraints to improve the functionality of generated sequences. Furthermore, addressing the computational efficiency of dWJS, particularly in terms of training time and resource consumption, will be important to make an impact in real-world protein design problems and pipelines. Future works can benefit from optimizing these aspects and exploring hybrid approaches that integrate inductive biases from more domain-specific knowledge to further improve the performance and generalizability of dWJS.

6 Conclusion

This project validates the performance of dWJS in antibody design, reaffirming its ability to generate diverse, functional, and high-quality protein sequences with remarkable computational efficiency. The findings underscore dWJS's potential to generalize beyond antibodies to other protein families, such as enzymes to advance innovations in protein engineering and synthetic biology. In this project, I replicated some of the findings of the authors of dWJS. I focused on hyperparameter sweeps for the optimal noise parameter for EBMs, assess protein quality in generated samples, evalued trade offs between learning rate and training time, and reproduced the robustness of the methods for discrete sequence generation.

Ongoing efforts to refine hyperparameters, optimize noise parameters, and enhance computational efficiency are important for the broader applicability of dWJS. In addition, addressing current limitations and incorporating domain-specific constraints will expand its utility across diverse protein design applications. By establishing dWJS as a robust and generalizable framework, this work lays the foundation for transformative advancements in generative protein design, addressing critical challenges in healthcare, bioengineering, and beyond.

References

Ethan C Alley, Grigory Khimulya, Surojit Biswas, Mohammed AlQuraishi, and George M Church. Unified rational protein engineering with sequence-based deep representation learning. *Nature Methods*, 16(12):1315–1322, 2019.

Prafulla Dhariwal and Alexander Quinn Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021.

Sander et al. Dieleman. Discrete walk-jump sampling: A framework for efficient protein design. *Advances in Neural Information Processing Systems*, 2023.

Emiel et al. Hoogeboom. Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances in Neural Information Processing Systems*, 34:12454–12465, 2021.

John et al. Ingraham. Generative models for graph-based protein design. *Advances in Neural Information Processing Systems*, 32, 2019.

John et al. Jumper. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873): 583–589, 2021.

Yann et al. LeCun. A tutorial on energy-based learning. *Predicting Structured Data*, 2006.

Ali et al. Madani. Large language models generate functional protein sequences across diverse families. *Nature Biotechnology*, 2023.

Alexander et al. Rives. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences*, 118(15): e2016239118, 2021.

Yang et al. Song. Score-based generative modeling through stochastic differential equations. *Advances in Neural Information Processing Systems*, 33:12438–12448, 2021.