SIPF: Sampling Method for Inverse Protein Folding

Tianfan Fu
Georgia Institute of Technology
Atlanta, GA, USA

Jimeng Sun
University of Illinois Urbana-Champaign
Urbana, IL, USA

ABSTRACT

Protein engineering has important applications in drug discovery. Among others, inverse protein folding is a fundamental task in protein design, which aims at generating protein's amino acid sequence given a 3D graph structure. However, most existing methods for inverse protein folding are based on sequential generative models and therefore limited in uncertainty quantification and exploration ability to the entire protein space. To address the issues, we propose a sampling method for inverse protein folding (SIPF). Specifically, we formulate inverse protein folding as a sampling problem and design two pretrained neural networks as Markov Chain Monte Carlo (MCMC) proposal distribution. To ensure sampling efficiency, we further design (i) an adaptive sampling scheme to select variables for sampling and (ii) an approximate target distribution as a surrogate of the unavailable target distribution. Empirical studies have been conducted to validate the effectiveness of SIPF, achieving 7.4% relative improvement on recovery rate and 6.4% relative reduction in perplexity compared to the best baseline.

CCS CONCEPTS

• **Computing methodologies** → *Continuous space search.*

KEYWORDS

protein design, sampling method, protein engineering, inverse protein folding, drug discovery

ACM Reference Format:
Tianfan Fu and Jimeng Sun. 2022. SIPF: Sampling Method for Inverse Protein Folding. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, Washington DC, USA, 11 pages. https://doi.org/10.1145/3534678.3539284

1 INTRODUCTION

Drug discovery is a highly complex optimization process that needs to fulfill multiple objectives. An ideal drug candidate should possess high target binding efficacy, balanced biophysical and biochemical properties, and low cytotoxicity [12, 18]. Biologics are an important class of therapeutics because of their high affinity, lower toxicity, and higher safety compared to small molecules. Eight of the top 10 best-selling drugs in 2018 were biologics [28]. Because biologics drugs are human-made proteins, protein engineering has attracted lots of attention for biologics development.

In many protein engineering applications, the desired 3D structure with a useful function was first observed before identifying the amino acid sequence. The task becomes designing an amino acid sequence that can properly fold into the desired 3D structure [20, 29, 41]. The problem is named *inverse protein folding*, which often requires machine learning models due to the complex nature of the task [20, 28, 41, 42]. Most existing approaches [20, 32, 37, 42] leverage autoregressive (sequential) generative model and learn a mapping from a 3D graph structure to an amino acid sequence. To predict the target amino acid sequence sequentially, different network architectures have been proposed to represent 3D protein structures, e.g., multiple structured transformers with multi-head self-attention components [20], three-dimensional convolutional neural network (3DCNN) [32, 42], graph convolutional network (GCN)[37]. However, several limitations (L1 and L2) remain in these existing methods. **(L1) Lack of uncertainty quantification in the protein space**: Generative models are mostly based on maximum likelihood learning, which uses point estimation and has difficulty in quantifying the uncertainty. **(L2) Incremental generation degradation during sequential generation**: these auto-regressive generative models generate amino acid sequences sequentially. The later amino acids depend heavily on the already generated ones thus, the error may accumulate during generation.

To alleviate these challenges, we proposed the Sampling method for Inverse Protein Folding (SIPF). The main contributions are: (1) **Uncertainty quantification** (address L1): To quantify uncertainty, we formulate inverse protein folding as a Markov Chain Monte Carlo (MCMC) sampling problem (Sec 3.1), where pretrained neural networks are used as MCMC proposal distribution (Sec 3.2), and an approximate target distribution is designed (Sec 3.4). (2) **Adaptive sampling** (address L2): we design an adaptive sampling method to sample more thoroughly at the variables[1] with high uncertainty. The designed sampler allows random and weighted scan over all the amino acids and provides more flexibility than the sequential generation (Sec 3.3). (3) **Theoretical results** guarantee that the SIPF's samples approximately follow the target distribution (Sec 3.6). We also conduct a thorough experiment to show the superiority of SIPF, which obtains 7.4% relative improvement on recovery rate and 6.4% relative reduction in perplexity (Sec 4).

2 RELATED WORK

Inverse Protein Folding. The target of inverse protein folding is to design an amino acid sequence that can properly fold into the input 3D structure [20, 29]. In recent years, deep learning methods have become state-of-the-art methods on inverse protein folding task. For example, [20] utilized multiple structured transformers

[1]In this paper, a variable corresponds to an amino acid.

with multi-head self-attention components; [32, 42] leveraged three-dimensional convolutional neural network (3DCNN); [37] transform a 3D graph structure (the target backbone protein structure) into a 2D graph (using adjacency matrix instead of 3D coordinates of nodes) and apply graph convolutional network (GCN) to obtain embeddings for nodes and edges; [7] jointly learns a sequence embedding using a transformer and a fold embedding from the density of secondary structural elements in 3D voxels. However, these generative models are mainly based on maximum likelihood learning and generate amino acid sequences in autoregressive manner, thus suffering from limitations as discussed (L1, L2 in Sec 1).

Protein Sequence Learning. There is also a parallel research line on (unconditional) amino acid sequence learning, including protein sequence representation learning [2, 5, 6, 17, 18, 26] and protein sequence generation [1, 21, 24, 27, 34, 36, 39]. The target of protein representation learning is to learn a semantically rich embedding for protein sequence, which leverage different kinds of deep learning models, e.g., Recurrent Neural Network (RNN) [2], Convolutional Neural Network (CNN) [18], bidirectional long short-term memory (bi-LSTM) enhanced by structural information [5]; self-supervised contrastive learning method [17, 26], Bidirectional Encoder Representations from Transformers (BERT) based pretraining approach [6]. On the other hand, deep learning models are also widely used in protein sequence generation, e.g., sequence level generative adversarial network (GAN) [34], sequence-level variational auto-encoder (VAE) [36], Long Short Term Memory (LSTM) [1], constitutive motifs level Restricted Boltzmann Machine [39], geometric graph neural network [21, 27], ensemble gradient method [24].

2.1 Background: Markov Chain Monte Carlo (MCMC)

Sampling methods (a.k.a., Bayesian sampling methods) are appealing in their ability to capture uncertainty and avoid overfitting, compared with maximum likelihood learning methods that leverage point estimation [25, 40]. Markov Chain Monte Carlo (MCMC) methods comprise a class of methods that sample from the target distribution [25]. Specifically, suppose we want to sample from the target distribution over x, denoted $P(x)$. MCMC constructs a Markov chain that has target distribution $P(x)$ as its equilibrium distribution. We briefly review three related MCMC methods: **(i) Metropolis-Hastings algorithm** is a mainstream MCMC approach [25]. The initialization (i.e., $x^{(0)}$) of Metropolis-Hastings algorithm can be a random point. At the t-th iteration, given the previous sample $x^{(t-1)}$, there are three steps: (1) generate *proposal* for the next iteration via $x' \sim q(\cdot|x^{(t-1)})$, where $q(\cdot|x^{(t-1)})$ is the MCMC proposal distribution, conditioned on $x^{(t-1)}$. A usual choice is to let $q(\cdot|x^{(t-1)})$ be a Gaussian distribution centered at $x^{(t-1)}$. (2) evaluate the acceptance rate

$$\mathcal{A}(x^{(t-1)} \rightarrow x') = \min\Big\{1, \frac{P(x')q(x^{(t-1)}|x')}{P(x^{(t-1)})q(x'|x^{(t-1)})}\Big\}. \quad (1)$$

(3) accept the *proposal* x' with probability $\mathcal{A}(x^{(t-1)} \rightarrow x')$. If accept, $x^{(t)} = x'$; otherwise, $x^{(t)} = x^{(t-1)}$. Metropolis-Hastings algorithm can be applied to any probability distribution. However, it suffers from slow mixing rate due to the random walk behaviour [25, 40]. For example, Gaussian based proposal distribution exhibited random walk beheviour, which is especially severe when x's dimension is high [25]. **(ii) Gibbs sampling** is another popular MCMC approach [13, 14]. Suppose the variable x is decomposed as $x = [x_1, \cdots, x_d]$, Gibbs sampling cycles through all the variables $x_1, \cdots, x_d$ and sample each one from its distribution conditioned on the current values of all other variables, for example, when sample i-th dimension, it uses the conditional distribution of x_i (conditioned on the remaining variables) $x_i \sim P_i(\cdot|\, x_{-i})$, where $x_{-i} = [x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_d]$ represents all other variables. Gibbs sampling suppresses the random-walk behaviour in Metropolis-Hastings algorithm and is more efficient, but it requires the conditional distribution $P_i(\cdot|x_{-i})$ to be analytically tractable, which is usually restrictive. **(iii) Metropolis-Hastings within Gibbs sampling (mixture of i and ii)** combines the advantage of Metropolis-Hastings algorithm and Gibbs sampling [25]. Specifically, like Gibbs sampling, it cycles through $x_1, \cdots, x_d$. Unlike Gibbs sampling, when conditional probability $P_i(x_i|\, x_{-i})$ is analytically intractable, it leverages Metropolis-Hastings algorithm to sample from $P_i(x_i|\, x_{-i})$. In this paper, we design a novel Metropolis-Hastings within Gibbs sampling method. The proposed method is different from traditional Metropolis-Hastings within Gibbs sampling method in the following two aspects: (1) our method cycles through all the variables with random order and adaptive weights so that we are able to sample more intensively at the variables with higher uncertainty. In contrast, existing methods cycle through all the variables in a deterministic order or random order with uniform weight. This aspect will be elaborated in Sec 3.3. (2) We design an approximate (biased) target distribution as a surrogate of the target distribution, since the target distribution is not available. A related method is pseudo-marginal MCMC [3], where the evaluation of the target distribution is replaced by its unbiased estimator. Pseudo-marginal MCMC is useful when the target density is not available analytically, e.g., latent variable models [3]. Differently, our method uses a biased estimator to estimate the target distribution, which is more challenging. It will be described in Sec 3.4.

3 SIPF METHOD

Overview. We describe SIPF in the following order: (1) **Problem formulation**: We describe the inverse protein folding task (Definition 1) and formulate it as a Markov Chain Monte Carlo (MCMC) sampling problem (Sec 3.1). (2) **MCMC proposal distribution** is a mixture of two pretrained neural networks: (i) Equivariant Graph Neural Networks (EGNN); (ii) Bidirectional Encoder Representations from Transformers (BERT) model (Sec 3.2). (3) **Adaptive sampling** is designed to adaptively assign more sampling budgets to the amino acids with more uncertainty (Sec 3.3). (4) **Approximate target distribution** $\widetilde{P}(\mathrm{S})$ is designed as surrogate of the unavailable target distribution to conduct Metropolis-Hastings algorithm (Sec 3.4). (5) **SIPF pipeline** (Sec 3.5). (6) **Theoretical analysis** guarantee that the SIPF's samples approximately follows the target distribution under certain assumptions (Sec 3.6).

3.1 Problem Formulation

This section formulates the inverse protein folding task. We start with some basic notations about protein structures. We represent a protein with three structures: a) Amino acid sequence (a.k.a. primary structure), b) secondary structure sequence, and c) a 3D graph structure. They describe the structure of a protein at different levels of complexity. Suppose we have N amino acids in a protein graph, each amino acid corresponds to a node in 3D graph structure, a token in the *amino acid sequence* and *secondary structure sequence*.

Table 1: Mathematical notations and descriptions.

Notations	Descriptions
N	number of amino acids in proteins.
V	set of all the amino acids.
g_i	coordinate of the i-th node in 3D graph $\mathcal{G}$.
$\mathcal{G} = (g_1, \cdots, g_N)$	3D graph structure (N nodes with coordinates).
$\mathbf{s}_i \in V$	The i-th amino acid in the sequence S.
$\mathrm{S} = (\mathbf{s}_1, \cdots, \mathbf{s}_N)$	Sequence of amino acid (length N).
$\mathrm{S}_{<i}$	the first $i-1$ amino acids in the sequence S.
$\mathrm{S}_{-i} = (\mathbf{s}_1, \cdots, \mathbf{s}_{i-1}, \mathbf{s}_{i+1}, \cdots, \mathbf{s}_N)$	amino acid sequence without i-th amino acid
$P(\mathrm{S}\vert\mathcal{G})$	target distribution of S given $\mathcal{G}$.
$Q_\theta(\mathbf{s}_i\vert\mathrm{S}_{-i}, \mathcal{G})$	MCMC proposal distribution, conditional prob. of $\mathbf{s}_i$
$\Vert z\Vert$	l_2 norm of the vector z
L_1	Number of layers in EGNN.
$\mathbf{m}_i^{(l)}$	message vector of node i at l-th layer;
$\mathbf{m}_{ij}^{(l)}$	message vector of edge from i to j at l-th layer;
$\mathbf{e}_i^{(l)}$	i-th node's embedding at l-th layer;
$\mathbf{x}_i^{(l)}$	i-th node's position embeddings at l-th layer;
$H_1(\cdot), H_2(\cdot), H_3(\cdot)$	embedding in BERT.
L_2	Number of transformer layers in BERT.
γ	hyperparameter Eq. 9
$\mathbf{1}(\cdot)$	Indicator function
$\mathrm{Distance}(\cdot, \cdot)$	Distance between two amino acid sequences (Eq. 12)
$\mathbf{q} = [q_1, \cdots, q_N]$	sampling weight for each amino acid, $\sum_{i=1}^N q_i = 1$.
$\lambda > 0$	hyperparameter in optimizing q (Eq. 14)
$\mathcal{A}(\mathrm{S} \rightarrow \mathrm{S}')$	acceptance rate from state S to S′ (Eq. 21)
$\widetilde{P}(\cdot)$	approximate target distribution (Eq. 18)

Definition 1 (Amino acid sequence (a.k.a. primary structure)). We use $\mathrm{S} = (\mathbf{s}_1, \cdots, \mathbf{s}_N)$ to denote the amino acid sequence, a.k.a. primary structure. The length is N, $\mathbf{s}_i$ represents the i-th token (i.e., amino acid) in the sequence. There are 20 categories of natural amino acids, e.g., Glycine, Valine, Leucine, etc. The list of all the amino acids and their frequencies is provided in Appendix. The set of all amino acids is denoted V, $|V| = 20$.

Definition 2 (Secondary structure sequence). We use $\mathbf{Z} = (\mathbf{z}_1, \cdots, \mathbf{z}_N)$ to denote the secondary structure sequence. The length is also N, $\mathbf{z}_i$ represents the kind of secondary structure that i-th amino acid belongs to. There are totally 8 categories of secondary structures, e.g., α-helix, Turn, Bend, π-helix, 3-10 helix, Strand, Isolated beta-bridge residue and None [28]. The list of all the secondary structures and their frequencies are provided in Appendix.

Definition 3 (3D graph structure of protein (a.k.a. tertiary structure)). We use $\mathcal{G} = (g_1, \cdots, g_N)$ to denote the 3D graph structure of the protein. There are N nodes in the graph, g_i represents the 3D coordinate of the i-th node. An amino acid consists of an α (central) carbon atom linked to an amino group, a carboxyl group, a hydrogen atom, and a variable component called a side chain. We use the position of α carbon atom as the coordinate of the amino acid, following [20, 21]. 3D graph structure is also known as the backbone structure, which belongs to the tertiary structure of a protein.

Problem 1 (Inverse Protein Folding). Given the 3D graph structure $\mathcal{G}$, inverse protein folding is to recover the amino acid sequence S. That is, our objective is to find an amino acid sequence $\mathrm{S} = (\mathbf{s}_1, \cdots, \mathbf{s}_N)$ ($\mathbf{s}_i \in V$) that maximize the likelihood function $P(\mathrm{S}|\mathcal{G})$,

$$\arg\max_{\mathrm{S}=(\mathbf{s}_1,\cdots,\mathbf{s}_N)} \; P(\mathrm{S}|\mathcal{G}), \tag{2}$$

Given the tertiary structure (i.e., 3D graph $\mathcal{G}$), the secondary structure $\mathbf{Z}$ is available [12]. Both $\mathcal{G}$ and $\mathbf{Z}$ are the input of the problem. Therefore, $P(\mathrm{S}|\mathcal{G}) = P(\mathrm{S}|\mathcal{G}, \mathbf{Z})$. We use $P(\mathrm{S}|\mathcal{G})$ for simplicity.

Existing methods mostly leverage deep generative models to generate the amino acid sequence in an auto-regressive way [7, 20, 32, 37, 42], where the likelihood function is decomposed as $P(\mathrm{S}|\mathcal{G}) = \prod_{i=1}^N P(\mathbf{s}_i|\mathrm{S}_{<i}, \mathcal{G})$, where $\mathrm{S}_{<i}$ denoted all the amino acids before the i-th amino acid, i.e., the first i-1 amino acids in the sequence. However, as mentioned in L1 and L2 in Sec 1, several limitations remain: (i) fails to capture uncertainty and (ii) incremental generation degradation during sequence generation.

Formulation: To tackle these issues, we formulate Problem (2) into a sampling problem, i.e., drawing samples from the *target distribution* $P(\mathrm{S} = (\mathbf{s}_1, \cdots, \mathbf{s}_N)|\mathcal{G})$. Our goal is to recover the amino acid as much as possible. We define recovery rate as the percentage of recovered amino acids,

$$\mathcal{R}(\mathrm{S}, \mathrm{S}_{\text{truth}}) = \Big(\sum\nolimits_{i=1}^{N} \mathbf{1}\big((\mathrm{S})_i = (\mathrm{S}_{\text{truth}})_i\big)\Big) / |\mathrm{S}_{\text{truth}}|, \tag{3}$$

where S and $\mathrm{S}_{\text{truth}}$ are the generated and groundtruth amino acid sequences respectively and have the same length. $(\mathrm{S})_i$ and $(\mathrm{S}_{\text{truth}})_i$ are the i-th amino acids in sequence S and $\mathrm{S}_{\text{truth}}$, respectively. $\mathbf{1}(\cdot)$ is the indicator function. The *target distribution* is defined as

$$P(\mathrm{S}|\mathcal{G}) \propto \exp(\delta \mathcal{R}(\mathrm{S}, \mathrm{S}_{\text{truth}})), \tag{4}$$

where $\delta > 0$ is the scaling hyperparameter. The target distribution $P(\mathrm{S}|\mathcal{G})$ is unavailable because the groundtruth amino acid sequence $\mathrm{S}_{\text{truth}}$ is unknown. Thus, it is hard to directly sample the target distribution using traditional Markov Chain Monte Carlo (MCMC) sampling methods (Sec 2.1).

3.2 MCMC Proposal distribution: Pretrained Neural Networks

This section describes MCMC proposal distribution $Q_\theta(\mathbf{s}_i|\mathrm{S}_{-i}, \mathcal{G})$, which is a mixture of two pretrained neural networks, including (1) equivariant graph neural network (geometric graph level); (2) BERT model (amino acid sequence level). Both models are pretrained in self-supervised manner [17]: predicting the category of masked node/token (i.e., amino acid) conditioned on the remaining variables (nodes/tokens/amino acids) and 3D graph structure.

3.2.1 Equivariant Graph Neural Network (EGNN) for 3D geometric graph. We leverage the state-of-the-art equivariant graph neural network (EGNN) proposed in [35]. It is translation-, rotation- and reflection- invariant with respect to an input set of 3D points. That is, the translation, rotation or reflection on the coordinates would not change the output. Node embeddings at the l-th layer are $\{\mathbf{e}_i^{(l)}\}_{i=1}^N$, where $l = 0, 1, \cdots, L_1$, L_1 is number of layers in EGNN. The initial node embeddings $\{\mathbf{e}_i^{(0)}\}_{i=1}^N$ embed the categories of amino acids and are randomly initialized. The target amino acid $(\mathbf{s}_i)$ is masked. Each kind of amino acid (including the masked one)

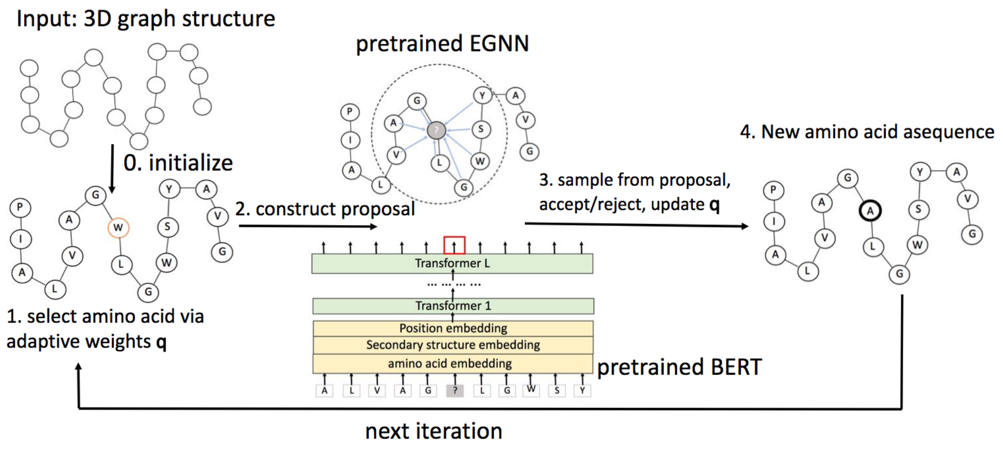
Input: 3D graph structure
0. initialize
1. select amino acid via adaptive weights **q**
2. construct proposal
pretrained EGNN
Transformer L
Transformer 1
Position embedding
Secondary structure embedding
amino acid embedding
pretrained BERT
3. sample from proposal, accept/reject, update **q**
4. New amino acid asequence
next iteration

Figure 1: The whole pipeline includes the following four key steps in a single iteration: (1) select amino acid (i.e., variable) based on adaptive weight q as described in Sec 3.3; (2) construct MCMC proposal distribution (masking the selected node and feeding the remaining structure into EGNN and BERT), which is a mixture of EGNN and BERT predictions as described in Sec 3.2; (3) sample from the MCMC proposal distribution, accept or reject the proposal, as described in Sec 3.4, update adaptive weight q (Sec 3.3). (4) generate the new amino acid sequence and jump to the next iteration.

corresponds to a node embedding. Coordinate embeddings at the l-th layer are denoted $\{\mathbf{x}_i^{(l)}\}_{i=1}^N$. The initial coordinate embeddings $\{\mathbf{x}_i^{(0)}\}_{i=1}^N$ are the real 3D coordinates of all the nodes, i.e., $\{g_i\}_{i=1}^N$. The following equation defines the update rule at the l-th layer ($l = 1, \cdots, L_1$):

$$\begin{aligned}\mathbf{m}_{ij}^{(l+1)} &= \text{MLP}_1\big(\mathbf{e}_i^{(l)}, \mathbf{e}_j^{(l)}, ||\mathbf{x}_i^{(l)} - \mathbf{x}_j^{(l)}||^2\big),\\ \mathbf{x}_i^{(l+1)} &= \mathbf{x}_i^{(l)} + \sum\nolimits_{j\neq i} \big(\mathbf{x}_i^{(l)} - \mathbf{x}_j^{(l)}\big)\text{MLP}_2\big(\mathbf{m}_{ij}^{(l)}\big),\\ \mathbf{m}_i^{(l+1)} &= \sum\nolimits_j \mathbf{m}_{ij}^{(l+1)}, \qquad \mathbf{e}_i^{(l+1)} = \text{MLP}_3\big(\mathbf{e}_i^{(l)}, \mathbf{m}_i^{(l+1)}\big),\end{aligned} \tag{5}$$

where $\text{MLP}_1(\cdot), \text{MLP}_2(\cdot), \text{MLP}_3(\cdot)$ are all two-layer multiple layer perceptrons (MLPs) with ReLU activation. Within the l-th layer, $\mathbf{m}_{ij}^{(l)}$ represent the message vector for the edge from node i to node j; $\mathbf{m}_i^{(l)}$ represents the message vector for node i, $\mathbf{x}_i^{(l)}$ is the position embedding for node i; $\mathbf{e}_i^{(l)}$ is the node embedding for node i. Suppose we are given a target node whose conditional probability must be evaluated as proposal distribution. We only consider its spatial k-nearest neighbors as the input graph structure of EGNN and omit the other nodes, following [20, 32]. Regarding edge (i, j), all the possible combinations of (i, j) are considered, which is equivalent to a fully-connected 3D graph. When the target node is $\mathbf{s}_i$, we attach an MLP structure to the last layer's node embedding of $\mathbf{s}_i$ to build EGNN proposal distribution,

$$\text{EGNN}(\mathbf{s}_i|\text{S}_{-i}, \mathcal{G}) = \text{MLP}_4(\mathbf{e}_i^{(L_1)}), \tag{6}$$

where $\text{MLP}_4(\cdot)$ is two-layer MLP with ReLU activation in the hidden layer and softmax activation in the output layer. The computational complexity of EGNN scales quadratically with respect to the number of nodes. Specifically, the complexity of computation of $\mathbf{m}_{ij}$ and $\mathbf{x}_i^{(l+1)}$ are both quadratic w.r.t. number of nodes (the first two lines in Eq. 5).

In summary, EGNN can leverage local geometric structure; however, due to its limited scalability, it cannot consider the global structure, i.e., all the amino acids in the protein. On the other hand, protein usually contains hundreds or thousands of amino acids where long-term dependency between amino acids is important [37]. To model this dependency, we use BERT below.

3.2.2 BERT model for amino acid sequence. Bidirectional Encoder Representations from Transformers (BERT) is a transformer-based pretraining technique for natural language processing (NLP) [8]. We utilize BERT to learn another MCMC proposal distribution, denoted $\text{BERT}(\mathbf{s}_t|\text{S}_{-t}, \mathcal{G})$. Different from EGNN that leverages local geometric structure, BERT is able to model long range dependency in the amino acid and secondary structure sequence. Specifically,

$$\begin{aligned}&\mathbf{v}_1^{(l+1)}, \cdots, \mathbf{v}_N^{(l+1)} = \text{Transformer}\big(\mathbf{v}_1^{(l)}, \cdots, \mathbf{v}_N^{(l)}\big),\\ &l = 0, \cdots, L_2, \quad \mathbf{v}_i^{(0)} = \big[H_1(\mathbf{s}_i) \oplus H_2(\mathbf{z}_i) \oplus H_3(i)\big],\end{aligned} \tag{7}$$

where $\oplus$ denotes the concatenation of vectors; $H_1(\mathbf{s}_i)$ is the amino acid level embedding to represent $\mathbf{s}_i$; $H_2(\mathbf{z}_i)$ is the secondary structure level embedding to represent $\mathbf{z}_i$, worth to mention that given the tertiary structure (3D graph $\mathcal{G}$), the secondary structure $\mathbf{Z}$ is available; $H_3(i)$ is the position embedding to represent the position i and capture the sequential nature of positions in vector space (i.e., $1, 2, 3, \cdots$) [6, 8]. These embeddings are concatenated to construct the basic (0-th layer) embeddings. Multiple transformer layers (with multi-head self-attention) are stacked to capture the long-range dependency. The proposal distribution of i-th node is produced by adding a two-layer MLP (softmax in output layer and ReLU in hidden layer) to the last layer's (L_2-th) node embedding $\mathbf{v}_i^{(L_2)}$,

$$\text{BERT}(\mathbf{s}_i|\text{S}_{-i}, \mathcal{G}) = \text{MLP}(\mathbf{v}_i^{(L_2)}). \tag{8}$$

3.2.3 Mixture of two proposal distribution. EGNN and BERT focus on local geometric structure and long-range dependency, respectively. To combine two proposal distribution (Eq. (6) and (8)), we use a linear interpolation to get a better proposal distribution,

$$Q_\theta(\mathbf{s}_i|\mathrm{S}_{-i},\mathcal{G}) = \gamma\ \mathrm{EGNN}(\mathbf{s}_i|\mathrm{S}_{-i},\mathcal{G}) + (1-\gamma)\ \mathrm{BERT}(\mathbf{s}_i|\mathrm{S}_{-i},\mathcal{G}), \tag{9}$$

where $0 < \gamma < 1$ is a hyperparameter that controls the weights of two proposal distributions.

3.3 Adaptive Sampling

This section designs an adaptive sampling scheme that selects $\mathbf{s}_1, \cdots, \mathbf{s}_N$ (amino acids) with adaptive weights. As mentioned in Sec 2.1, one advantage of MCMC sampling methods over the autoregressive generative models [20, 32, 37] is that MCMC methods (especially Gibbs sampling) can sample variables (amino acids here) in a random order [14, 16], whereas, auto-regressive models produce variables sequentially in a fixed order.

Moreover, most proteins exhibit large variability in only a small fraction of the amino acids [9]. For example, in Y-shaped antibodies, complementarity-determining regions are part of the highly diverse chains, which only consist of a small fraction of the whole antibodies' amino acid sequence. In contrast, the remaining regions have significantly smaller variability [9]. We hope to assign more sample weight to the positions with high variability to explore the protein space efficiently and thoroughly. In particular, we expect the sampling methods to draw less "correlated" (more independent) samples. The independence between consecutive MCMC samples is usually measured by Effective Sample Size (ESS) [10, 25, 31]. The ESS is the number of effective independent samples from the target distribution to which the Markov chain is equivalent. ESS is defined as

$$\mathrm{ESS} = T/\big(1 + 2\sum\nolimits_{k=1}^{\infty}\rho_k\big), \tag{10}$$

where T is the number of samples, ρ_k is the auto-correlation of the sampler with lag k and measure correlation between i-th and $(i+k)$-th samples ($i = 1, 2, \cdots$). $\big(1 + 2\sum_{k=1}^{\infty}\rho_k\big)^{-1}$ is also known as the asymptotic efficiency of an MCMC sampler [25]. To explore the amino acid sequence space thoroughly, our goal is to maximize ESS given a fixed computational budget (i.e., T), which is equivalent to minimizing $\sum_{k=1}^{\infty}\rho_k$.

However, the higher-order autocorrelation coefficient (ρ_k) is difficult to estimate. Following [31], we minimize ρ_1 instead. Minimizing ρ_1 is equivalent to maximize the expected distance between consecutive samples (EDCS), i.e., $\mathrm{S}^{(t)}$ (t-th sample) and $\mathrm{S}^{(t+1)}$ ((t+1)-th sample), defined as ($\mathcal{E}$ denotes expectation)

$$\underset{\mathbf{q}=[q_1,\cdots,q_N]}{\arg\max}\ \ \mathrm{EDCS}(\mathbf{q}) = \mathcal{E}_{\mathbf{q}}\big[\mathrm{Distance}(\mathrm{S}^{(t+1)},\mathrm{S}^{(t)})\big], \tag{11}$$

where q_i is the sampling probability for the i-th amino acid $\mathbf{s}_i$ and we have $\sum_{i=1}^{N} q_i = 1$, $\mathrm{S}^{(t)}$ represents sampled amino acid sequence at the t-th iteration and $t < T$, T is the total number of samples. In this paper, during sampling process, two amino acid sequences $(\mathrm{S}_1, \mathrm{S}_2)$ are based on the same 3D graph structure $\mathcal{G}$ (with N nodes), the distance between S_1 and S_2 is defined as the total number of non-equal amino acids at all the corresponding positions,

$$\mathrm{Distance}(\mathrm{S}_1,\mathrm{S}_2) = \sum\nolimits_{i=1}^{N}\mathbf{1}\big((\mathrm{S}_1)_i \neq (\mathrm{S}_2)_i\big). \tag{12}$$

where $(\mathrm{S}_1)_i$ and $(\mathrm{S}_2)_i$ are the i-th amino acids in sequence S_1 and S_2, respectively, the indicator function $\mathbf{1}(\cdot)$ is equal to 0 when two amino acids are same and 1 otherwise. It is intractable to decompose EDCS in Eq. (11) due to the interaction between variables. To circumvent this issue, we make mean-field approximation as follows.

Assumption 1 (Mean-field approximation)**.** The variables $(\mathbf{s}_1, \cdots, \mathbf{s}_N)$ in the target distribution $P(\mathrm{S}|\mathcal{G})$ (Eq. 2) are independent from each other. That is, $P(\mathrm{S}|\mathcal{G}) = \prod_{i=1}^{N} P(\mathbf{s}_i|\mathcal{G})$. It is commonly used in the sampling problem, especially variational inference [25].

Under the mean-field approximation, $\mathbf{s}_1, \cdots, \mathbf{s}_N$ in the target distribution $P(\mathrm{S}|\mathcal{G})$ (Eq. 2) are independent from each other. EDCS (Eq. 11) can be decomposed as

$$\mathrm{EDCS}(\mathbf{q}) = \sum\nolimits_{i=1}^{N} q_i D(i),$$

$$D(i) = \sum_{j=1}^{|V|}\underbrace{p_j^i}_{\text{start at } j}\cdot\big(\underbrace{p_j^i\cdot 0}_{j\longrightarrow j} + \underbrace{(1-p_j^i)\cdot 1}_{j\longrightarrow\text{not } j}\big) = 1 - \sum_{j=1}^{|V|}(p_j^i)^2, \tag{13}$$

where $p_j^i = P(\mathbf{s}_i = j|\mathcal{G})$ is the MCMC proposal distribution under mean-field approximation, and $\sum_{j=1}^{|V|} p_j^i = 1$. V is set of all amino acids and $|V| = 20$. (1) "start at j": p_j^i is the probability that the current $\mathbf{s}_i$'s state is j. Due to the mean-field approximation, it is only related to the marginal distribution of $\mathbf{s}_i$, $P(\mathbf{s}_i = j|\mathcal{G})$. (2) "$j \longrightarrow j$": with probability p_j^i, $\mathbf{s}_i$ remains the same after sampling, which is not related to the starting point. In this case, the distance between two sequences before and after sampling is 0. (3) "$j \longrightarrow$ not j": with probability $1 - p_j^i$, the amino acid at $\mathbf{s}_i$ changes after sampling. In this case, the distance between two sequences before and after sampling is 1 based on distance function in Eq. (12). $D(i)$ measures the expectation of distance between consecutive sequences when sampling the i-th amino acid under the mean-field assumption. $D(i)$ can be estimated empirically during the sampling process. Now our objective becomes

$$\underset{\mathbf{q}=[q_1,\cdots,q_N]}{\arg\max}\ \sum\nolimits_{i=1}^{N} q_i D(i) + \lambda\frac{1}{N}\ln q_i,\ \text{ s.t. } \sum\nolimits_{i=1}^{N} q_i = 1, \tag{14}$$

where $\lambda > 0$ is a hyperparameter. The first term aims to maximize the expected distance between consecutive samples (EDCS); while the second term is the negative cross-entropy loss between the uniform distribution $[1/N, \cdots, 1/N]$ and adaptive weight's distribution $[q_1, \cdots, q_N]$, which serves as a regularizer that encourages the adaptive weight's distribution to be close to uniform distribution and encourages the coverage of all the variables during the sampling process. Only optimizing the first term would result in only sampling the variable with maximal $D(i)$ ($q_i = 1$). On the other hand, only optimizing the second term would lead to a uniformly random sampling with respect to all the variables. Eq. (14) is a constrained concave optimization problem. The constraints are a convex hull. We leverage the online augmented Lagrangian method to solve it in online manner [23].

Intuition. We observe that when the amino acids' distribution is more uniform, based on Eq. 14, larger $D(i)$ leads to larger q_i, i.e., larger sampling probability at i-th variable. This is consistent with our intuition: to explore the data space more thoroughly, we assign more weight to the amino acids that own significant uncertainty.

Then we show the effectiveness of the adaptive sampling scheme, i.e., the stationary distribution of the generated samples converges to the target distribution. The proof is in Appendix.

Theorem 1. The stationary distribution of the samples produced by adaptive sampling (using adaptive weight $\mathbf{q} = [q_1, \cdots, q_N]$ to select $\mathbf{s}_i$ to sample) is the expected target distribution $P(\mathrm{S}|\mathcal{G})$ (Eq. 4).

3.4 Approximate Target Distribution

This section designs an *approximate target distribution* $\widetilde{P}(\mathrm{S})$ as surrogate of the unavailable target distribution $P(\mathrm{S}|\mathcal{G})$. Specifically, as mentioned in Eq. (4), the target distribution $P(\mathrm{S}|\mathcal{G})$ is not available. To address this issue, we resort to *approximate target distribution*,

$$\widetilde{P}(\mathrm{S}) \approx P(\mathrm{S}|\mathcal{G}). \tag{15}$$

It is a function of the perplexity score of the amino acid sequence. Perplexity is commonly used in natural language processing for evaluating contextual representations [11] and is the exponential of the average log-likelihood and measures how well a probability model predicts a sequence of amino acids [4, 20], defined as

$$\text{Perplexity}(\mathrm{S}) = \exp\big(-(1/N) \sum\nolimits_{i=1}^{N} \log O(\mathbf{s}_i|\mathrm{S}_{<i}) \big). \tag{16}$$

$O(\cdot|\mathrm{S}_{<i})$ is a categorical distribution over all the amino acids conditioned on $\mathrm{S}_{<i}$, $O(\mathbf{s}_i|\mathrm{S}_{<i})$ measures the probability of $\mathbf{s}_i$, ranging from 0 to 1. It is usually a well trained sequence labelling model, e.g., LSTM [11]. Lower perplexities indicate better performance.

Existing studies empirically showed that perplexity score is negatively correlated to the recovery rate (Eq. 3) in the inverse protein folding [20, 21], which is also empirically validated in Sec 4.4. Formally, we assume

$$\mathcal{R}(\mathrm{S}, \mathrm{S}_{\text{truth}}) = \xi_1 - \xi_2 \text{Perplexity}(\mathrm{S}) + \epsilon. \tag{17}$$

where $\xi_1, \xi_2 > 0$ are coefficients and are empirically estimated on the validation set; ϵ is the bias and can be seen as zero-mean white noise. As mentioned in Eq. (4), the target distribution is defined as a function of recovery rate, i.e., $P(\mathrm{S}|\mathcal{G}) \propto \exp(\delta \mathcal{R}(\mathrm{S}, \mathrm{S}_{\text{truth}}))$. Combining Eq. (17) and (4), we get a biased estimator of the target distribution $P(\mathrm{S}|\mathcal{G})$ (Eq. 4) based on the perplexity score,

$$\begin{aligned} \widetilde{P}(\mathrm{S}) \;\; &\propto \;\; \exp\big(\delta(\xi_1 - \xi_2\text{Perplexity}(\mathrm{S}) + \epsilon)\big) \\ \approx \exp\big(\delta(\xi_1 &- \xi_2\text{Perplexity}(\mathrm{S}))\big) \propto \exp\big(-\eta \text{Perplexity}(\mathrm{S})\big), \end{aligned} \tag{18}$$

where $\delta > 0$ is the scaling hyperparameter defined in Eq. (4). We only need to tune the hyperparameter $\eta = \delta\xi_2$ on validation set.

3.5 SIPF Pipeline

This section summarizes the SIPF pipeline, especially how to incorporate Sec 3.2-3.4 into the sampling pipeline. As mentioned in Sec 3.1, SIPF is a novel Metropolis-Hastings within Gibbs sampling method ((iii) in Sec 2.1). Each single sampling iteration conducts Metropolis-Hastings step ((i) in Sec 2.1) on one amino acid. Specifically, before conducting the t-th step, we have amino acid sequence at (t-1)-th step,

$$\mathrm{S}^{(t-1)} = (\mathbf{s}_1^{(t-1)}, \cdots, \mathbf{s}_N^{(t-1)}). \tag{19}$$

Then based on adaptive weight $\mathbf{q}$ (Sec 3.3), we select to sample the $j^{(t)}$-th amino acid at the t-th step. Then we sample $\mathbf{s}_{j^{(t)}}^{(t)}$ from the MCMC proposal distribution $Q_\theta(\cdot|\mathrm{S}_{-j^{(t)}}^{(t-1)}, \mathcal{G})$ (Sec 3.2). In the *proposal* S', we update $\mathbf{s}_{j^{(t)}}^{(t)}$ while other amino acids remain the same, i.e.,

$$\begin{aligned} &\mathrm{S}' = (\mathbf{s}_1^{(t-1)}, \cdots, \mathbf{s}_{j^{(t)}-1}^{(t-1)}, \mathbf{s}_{j^{(t)}}^{(t)}, \mathbf{s}_{j^{(t)}+1}^{(t-1)}, \cdots, \mathbf{s}_N^{(t-1)}), \\ &\text{where } \mathbf{s}_{j^{(t)}}^{(t)} \sim Q_\theta(\cdot \mid \mathrm{S}_{-j^{(t)}}^{(t-1)}, \mathcal{G}). \end{aligned} \tag{20}$$

Different from the standard Metropolis-Hasting algorithm (Eq. 1, Sec 2.1), the acceptance rate uses approximate target probability $\widetilde{P}(\cdot)$ (Sec 3.4) instead of target probability,

$$\mathcal{A}(\mathrm{S}^{(t-1)} \to \mathrm{S}') = \min\left\{1, \frac{\widetilde{P}(\mathrm{S}')Q_\theta\big(\mathbf{s}_{j^{(t)}}^{(t-1)}|\mathrm{S}_{-j^{(t)}}^{(t)}, \mathcal{G}\big)}{\widetilde{P}(\mathrm{S})Q_\theta\big(\mathbf{s}_{j^{(t)}}^{(t)}|\mathrm{S}_{-j^{(t)}}^{(t-1)}, \mathcal{G}\big)}\right\}, \tag{21}$$

where in denominator and numerator in RHS, the proposal distribution are from the same distribution (i.e., $Q_\theta\big(\cdot|\mathrm{S}_{-j^{(t)}}^{(t-1)}, \mathcal{G}\big)$), because the $j^{(t)}$-th node is masked when generating proposal distribution (Sec 3.2), and $\mathrm{S}_{-j^{(t)}}^{(t-1)} = \mathrm{S}_{-j^{(t)}}^{(t)}$ (Eq. 19, 20). When the proposal is accepted, we set $\mathrm{S}^{(t)} = \mathrm{S}'$; otherwise, we use the sequence at (t-1)-th step, i.e., $\mathrm{S}^{(t)} = \mathrm{S}^{(t-1)}$.

Algorithm 1 SIPF

```
1: Input: train data $\{\mathcal{G}, \mathrm{S}, \mathrm{Z}\}$; test data (graph) $\{\mathcal{G}, \mathrm{Z}\}$.
2: Output: test data (amino acid sequence) $\{\mathrm{S}\}$.
3: Pretrain EGNN (Sec 3.2.1) and BERT (Sec 3.2.2) on train data
4: Random initialize $\mathrm{S}^{(0)}$ on test data.
5: for $t = 1, \cdots, T$ do
6:     Select $j^{(t)}$ based on adaptive weight $\mathbf{q} = [\mathbf{q}_1, \cdots, \mathbf{q}_N]$.
7:     Sample $\mathbf{s}_{j^{(t)}}$ from MCMC proposal distribution and generate
       the proposal $\mathrm{S}'$ (Eq. 20); update $D(j^{(t)})$ (Eq. 13)
8:     Estimate $\mathbf{q} = [q_1, \cdots, q_N]$ (Eq. 14) if (t=0 mod K).
9:     Evaluate the acceptance rate $\mathcal{A}(\mathrm{S}^{(t-1)} \to \mathrm{S}')$ (Eq. 21)
10:    Accept proposal with probability $\mathcal{A}(\mathrm{S}^{(t-1)})$, i.e., $\mathrm{S}^{(t)} = \mathrm{S}'$;
       otherwise, reject proposal, i.e., $\mathrm{S}^{(t)} = \mathrm{S}^{(t-1)}$.
11: end for
```

Explanation of Algorithm 1. We summarize key steps in Algorithm 1. First, we pretrain EGNN and BERT as conditional probability based MCMC proposal distribution using the training data (Step 3-4). Then during inference phase, within each sampling iteration, we conduct several steps: (i) Select a variable to sample and sample from MCMC proposal distribution, generate proposal S' (Step 7-8). (ii) update adaptive weight $\mathbf{q} = [q_1, \cdots, q_N]$ (Eq. 14) every K iterations (Step 8). (iii) accept/reject proposal S' using approximate target distribution and Metropolis-Hastings method (Step 9-10).

3.6 Theoretical Analysis

Now we explore the theoretical properties of the proposed method. We have two main theoretical results: (1) Lemma 1 and Theorem 2 shows the samples generated by our method follow the approximate target distribution $\widetilde{P}(\mathrm{S})$ (Eq.18); (2) we show the approximate target distribution $\widetilde{P}(\mathrm{S})$ is close to the target distribution $P(\mathrm{S}|\mathcal{G})$ (Eq.4) under certain assumptions. Proof are in Appendix.

Lemma 1. In Algorithm 1, the Markov chain of the sampled amino acid sequences $(\mathrm{S}^{(1)}, \mathrm{S}^{(2)}, \cdots)$ is ergodic over the approximate target distribution $\widetilde{P}(\mathrm{S})$.

Theorem 2. $\widetilde{P}(\mathrm{S})$ (Eq. 18) is maintained as the invariant distribution for the whole Markov chain produced by Algorithm 1.

Remarks. Lemma 1 and Theorem 2 guarantee SIPF's samples follow the approximate target distribution $\widetilde{P}(\mathrm{S})$.

The second part of theoretical analysis is to show the approximate target distribution $\widetilde{P}(\mathrm{S})$ is close to the target distribution $P(\mathrm{S}|\mathcal{G})$ (Eq. 4). Specifically, KL divergence is widely used to measure the difference between two probability distributions, e.g., in variational inference [25]. Thus, we attempt to prove that the KL divergence between the target probability $P(\mathrm{S}|\mathcal{G})$ and the approximate probability $\widetilde{P}(\mathrm{S})$ can be bounded. Before that, we formally define the KL divergence and make some assumptions.

Definition 4 (KL divergence). If we have two separate probability distributions $p(x)$ and $q(x)$ over the same random variable x, we can measure how different these two distributions are using the Kullback-Leibler (KL) divergence: $\mathrm{KL}(p(x)\|q(x)) = \mathcal{E}_{x\sim p}\big[\ln\frac{p(x)}{q(x)}\big] = \int p(x)\ln\frac{p(x)}{q(x)}dx$. The KL divergence of any two probability distributions $p(x)$ and $q(x)$ is greater or equal to 0. The equality is obtained if and only if $p = q$ almost everywhere. Lower KL divergence indicates these two distributions are closer.

Assumption 2. The negative correlation between the recovery rate and perplexity score is well estimated: given ξ_1, ξ_2, ϵ in Eq.(17), we have $|\xi_1 - \xi_1^{\mathrm{truth}}| < \tau_1$, $|\xi_2 - \xi_2^{\mathrm{truth}}| < \tau_2$, the white noise ϵ satisfies $|\epsilon| < \tau_3$, $\xi_1^{\mathrm{truth}}, \xi_2^{\mathrm{truth}}$ are real underlying coefficients.

Assumption 3. Perplexity of the generated amino acid sequences is bounded by ρ. It is validated by experiment in Section 4.4

Theorem 3. Under Assumption 2 and 3, the KL divergence between the target distribution and the approximate target distribution can be bounded as $\mathrm{KL}(P(\mathrm{S}|\mathcal{G})\|\widetilde{P}(\mathrm{S})) \leq \delta(\tau_1 + \tau_2\rho + \tau_3)$, where δ is scaling parameter in Eq.(4); τ_1, τ_2, τ_3 are defined in Assumption 2, ρ is defined in Assumption 3.

Remarks. The theorem shows the approximate target distribution is close to the target distribution in terms of KL divergence.

4 EXPERIMENT

4.1 Dataset and Preprocessing

RCSB. We download all the protein data in pdb format from https://www.rcsb.org/. The Protein Data Bank (PDB) file format is a textual file format describing the three-dimensional structures of molecules held in the Protein Data Bank. The PDB format accordingly provides description and annotation of protein and nucleic acid structures, including atomic coordinates, secondary structure assignments, and atomic connectivity. The list of all the amino acids, secondary structure, and their frequencies are provided in Appendix. We collect 27,043 proteins with a single chain, from which we randomly select 1,000 proteins as a test set. The remaining proteins are used for learning. We split training and validation sets with a 9:1 ratio. The training and validation set contains 24,338 and 2,705 proteins, respectively.

CATH [30] is a dataset based on the hierarchical classification of protein structure (CATH) available at https://www.cathdb.info/, also in PDB format. Following [7, 19, 20], for all domains in the CATH 4.2 40% non-redundant set of proteins, we collect full chains up to length 500 and then randomly assign their CATH topology classifications (CAT codes) to train, validation and test sets at a targeted 8/1/1 split. Since each chain can contain multiple CAT codes, we first removed any redundant entries from train and then from validation. Finally, we removed any chains from the test set that had CAT overlap with train and removed chains from the validation set with CAT overlap to train or test. This resulted in a dataset of 15,802 chains in the training set, 1,975 chains in the validation set, and 1,887 chains in the test set.

4.2 Baseline

For all baselines, we use the default setup (hyperparameter) in the original papers. (i) **StructTrans (Structured Transformer)** [20] uses three layers of self-attention and position-wise feedforward modules for the encoder and decoder; (ii) **ProDCoNN (Protein design convolutional neural network)** [42] uses a gridded box centered on the target residue to capture the local structural information. The atoms and their features are later voxelized into the 3D voxel grid. A 3D convolutional layer followed by a max-pooling layer is then attached, followed by a MLP layer to make a prediction; (iii) **DeepGCN (Deep Graph Convolutional Network)** [37] used graph convolutional network to represent the node and edge attributes, where the 3D graph is transformed into a 2D graph with an adjacency matrix. (iv) **Fold2Seq (Protein Folding to Sequence)** [7] jointly learns a sequence embedding using a transformer and a fold embedding from the density of secondary structural elements in 3D voxels. Traditional physics based method RosettaDesign [22] performs much worse than state-of-the-art deep learning methods and is inefficient [7, 20, 42, 44]. Thus, it is not included in the baselines. For reference, we also show the results of (i) **Uniform (Uniform frequencies)**: random amino acid sequence under the uniform distribution of all the amino acids and (ii) **Natural (Natural frequencies)**: random amino acid sequence through natural frequencies of amino acids. We calculate the natural frequencies of all the amino acids on the processed protein data and report them in Appendix.

4.3 Evaluation Metrics

We use evaluation metrics following [20, 32, 37, 42]. **(1) Recovery rate (RR)** (%): percentage of correctly recovered amino acids in the whole sequence; **(2) Perplexity (PPL)** has been defined in Eq.(16) and measures how well a probability model can predict a protein. Lower perplexities indicate better performance. **(3) Amino acid level accuracy (AAA)**. For each amino acid, the prediction can be seen as a binary classification task (correctly recovered or not), we report average Precision-Recall Area Under the Curve (PR-AUC) over all the amino acids to measure amino acid level accuracy.

4.4 Results and Analysis

In this section, we report the experimental results and analyze the results. The performance of all the compared methods on RCSB and CATH are presented in Table 2. We observe that our method

Table 2: Experimental results on RCSB and CATH. The results are averages and standard deviations of 5 independent runs. On each metric, we highlight the best score and use * to denote the results pass the t-test (SIPF versus Fold2Seq, the best baseline) with p-value < 0.05. The t-test results show that improvements of SIPF over the best baseline method are significant in most of the metrics on both tasks.

	Method	RR (↑)	PPL (↓)	AAA (↑)
RCSB	Uniform	5.52±0.13%	20.02±0.06	0.15±0.01
	Natural	9.18±0.08%	17.44±0.07	0.21±0.01
	StructTrans	29.81±0.15%	9.30±0.11	0.40±0.02
	ProDCoNN	25.78±0.25%	9.92±0.21	0.35±0.02
	DeepGCN	28.00±0.24%	9.67±0.11	0.38±0.01
	Fold2Seq	30.20±0.24%	9.28±0.10	0.43±0.01
	SIPF	**32.43±0.23%***	**8.69±0.13***	**0.46±0.01***
	Method	RR (↑)	PPL (↓)	AAA (↑)
CATH	Uniform	5.13±0.04%	20.03±0.05	0.15±0.01
	Natural	9.84±0.05%	17.50±0.04	0.20±0.01
	StructTrans	28.56±0.08%	9.47±0.06	0.38±0.01
	ProDCoNN	26.52±0.11%	9.85±0.10	0.36±0.01
	DeepGCN	27.78±0.12%	9.71±0.11	0.38±0.01
	Fold2Seq	30.02±0.11%	9.38±0.05	0.43±0.01
	SIPF	**31.45±0.13%***	**8.72±0.10***	**0.45±0.01**

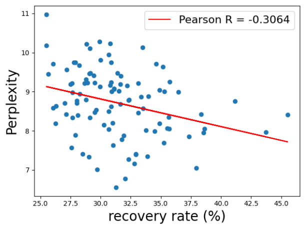

Figure 2: The correlation between perplexity and amino acid level recovery rate. Perplexity and recovery rate are negatively correlated.

achieves the highest recovery rate (RR), amino acid-level accuracy (AAA), and lowest (best) perplexity among all the compared methods on both datasets. Specifically, compared with the best baseline method Fold2Seq, our method achieves 7.4% relative improvement on recovery rate (RR) (30.20% v.s. 32.43%) and 6.4% relative reduction in perplexity (8.69 v.s. 9.28) on RCSB, 4.5% relative improvement on recovery rate (RR) (30.02% v.s. 31.45%) and 7.0% relative reduction in perplexity (8.72 v.s. 9.38) on CATH. The results of hypothesis testing (t-test) shows the improvements over the best baseline method are significant in most of the metrics.

We also plot the scatter points of perplexity and amino acid level recovery rate on a subset of the test set of RCSB in Fig. 2 to reveal their correlation. Their Pearson (product-moment) correlation coefficient is -0.3064, indicating that perplexity and recovery rate are negatively correlated. This observation is consistent with the existing studies [20, 21] and validates the rationality of incorporating perplexity into the approximate distribution $\widetilde{P}(\mathbf{S})$ (in Eq.18).

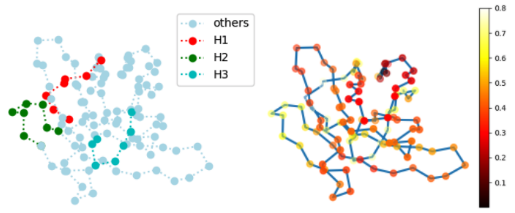

Figure 3: Case study: visualizing (normalized) adaptive weight (q, Eq. 14). Left: 3D structure of the heavy chain of the protein whose PDB id is 4BKL, where H1, H2, H3 usually have high variability compared with other positions. Right: 3D structure of the heavy chain with normalized sampling weight q at each node (amino acid). Lighter nodes have a higher weight. We find amino acids on H1, H2, H3 have more weights than others, consistent with the existing knowledge that these regions have higher variability [38, 43].

4.5 Case Study

As described in Sec 3.3, our method estimates a sampling weight q_i for the i-th amino acid. The amino acid with more uncertainty will have a higher sampling weight. To get more insight into the proposed method, we visualize normalized $\mathbf{q}$ weight on a data sample as a case study in Figure 3. Specifically, we select an antibody whose pdb id is 4BKL as the example. Detailed description of 4BKL is available at https://www.rcsb.org/structure/4bkl. Antibody is a special kind of protein, which has a symmetric Y shape, each half of the symmetric unit has two chains: a heavy chain (H) and a light (L) chain [38, 43]. There are 6 open loops, L1, L2, L3 on the light chain and H1, H2, H3 on the heavy chain, named complementarity determining regions, which define most of their antigen binding functionality. Thus, these 6 loops usually have higher variability [38, 43]. In this section, for ease of visualization, we show the 3D structure of the heavy chain of 4BKL in Figure 3, including the positions of H1, H2, H3 loops (left) and the estimated sampling weight $\mathbf{q}$ (right), where lighter nodes in the right sub-figure mean higher weight. The sampling weights are normalized to range from 0 to 1. We plot the 3D structure of the heavy chain with normalized (range from 0-1) sampling weight $\mathbf{q}$ at each node (amino acid). We observe that the nodes (i.e., amino acid) on H1, H2, H3 have more weights than other nodes, which is consistent with the fact that these regions have higher variability [38].

Table 3: Ablation studies (Sec 4.6).

Method	RR (↑)	PPL (↓)	AAA (↑)
EGNN only	32.17±0.23%	8.71±0.13	0.45±0.01
BERT only	29.80±0.27%	9.47±0.11	0.40±0.01
Gibbs sampling	28.81±0.14%	9.62±0.12	0.39±0.02
uniform sampling	31.50±0.15%	9.28±0.12	0.42±0.02
w.o. reject	31.75±0.21%	9.20±0.16	0.43±0.02
SIPF	**32.43±0.23%**	**8.69±0.13**	**0.46±0.01**

4.6 Ablation Study

To further understand our method, we conduct an ablation study on the RCSB dataset to investigate the impact of each component on the performance. Specifically, we explore the empirical effect for both conditional probability based proposal and sampling method and consider the following variants of our method. **(1) EGNN only**. Our MCMC proposal distribution is a mixture of EGNN and BERT prediction, as described in Eq.(9). The variant uses only EGNN prediction as a proposal to consider the local geometric structural information only, i.e., $\gamma = 1$ (Eq.9 in Sec 3.2). **(2) BERT only**. The variant uses only BERT prediction as MCMC proposal distribution to consider the long-range dependency only, i.e., $\gamma = 0$ (Eq.9 in Sec 3.2). **(3) Gibbs sampling**. Gibbs sampling scans all the variables in a fixed order [13], instead of adaptive sampling in our method (Sec 3.3). **(4) uniform sampling**. The variant randomly and uniformly selects all the variables instead of adaptive weight (Sec 3.3). **(5) w.o. reject**. The variant does not reject the proposal but accepts all the proposals. It studies the effect of leveraging approximate target distribution (Sec 3.4). To make the comparison fair, the total numbers of sampling iterations for all the methods are the same, ten times of the amino acid sequence length. Table 3 reports the results of the ablation study, which demonstrates the best performance of the full method SIPF. From the first two lines, we find that both EGNN and BERT have positive contributions to the performance. We observe that removing EGNN causes most degradation in both recovery rate and perplexity, suggesting that EGNN is more important than BERT. This is also validated by the fact that $\gamma = 0.7$ achieves the best performance by putting more weight on the EGNN component. Comparing 3rd, 4th and the last line, we also observe that adaptive sampling in SIPF outperforms Gibbs sampling and uniform sampling. In addition, comparing the last two lines, we find if we accept the proposal, the performance will degrade, demonstrating the positive contribution of approximate target distribution $\widetilde{P}(\mathbf{S})$ (Sec 3.4). In sum, MCMC proposal, adaptive sampling, and approximate target distribution are all key components of SIPF.

5 CONCLUSION

In this paper, to address the challenges in the existing inverse protein folding methods, we have proposed a sampling based method for inverse protein folding. Concretely, we first formulate it as a sampling problem and then design two pretrained neural networks as (conditional probability) MCMC proposal distribution. We also design a novel sampling method (an adaptive sampling scheme and approximate target distribution) to quantify uncertainty and enhance exploration ability to data space. Theoretical results show the SIPF's samples approximately follows the target distribution. Thorough empirical studies confirm SIPF's superiority.

ACKNOWLEDGEMENTS

This work was supported by IQVIA, NSF award SCH-2014438, IIS-1838042, NIH award R01 1R01NS107291-01 and OSF Healthcare. We thank Cao Xiao, Wenhao Gao and Xinyu Gu for discussions.

REFERENCES

[1] Rahmad Akbar et al. 2021. In silico proof of principle of machine learning-based antibody design at unconstrained scale. *BioRXiV* (2021).
[2] Ethan C Alley et al. 2019. Unified rational protein engineering with sequence-based deep representation learning. *Nature methods* (2019).
[3] Christophe Andrieu and Gareth O Roberts. 2009. The pseudo-marginal approach for efficient Monte Carlo computations. *The Annals of Statistics* (2009).
[4] Jose Juan Almagro Armenteros et al. 2020. Language modelling for biological sequences–curated datasets and baselines. *BioRxiv* (2020).
[5] Tristan Bepler and Bonnie Berger. 2019. Learning protein sequence embeddings using information from structure. *ICLR* (2019).
[6] Nadav Brandes et al. 2021. ProteinBERT: A universal deep-learning model of protein sequence and function. *bioRxiv* (2021).
[7] Yue Cao et al. 2021. Fold2Seq: A Joint Sequence (1D)-Fold (3D) Embedding-based Generative Model for Protein Design. In *ICML*.
[8] Jacob Devlin et al. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *NAACL* (2019).
[9] Mathieu Dondelinger et al. 2018. Understanding the significance and implications of antibody numbering and antigen-binding surface/residue definition. *Frontiers in immunology* (2018).
[10] Tianfan Fu et al. 2020. MIMOSA: Multi-constraint Molecule Sampling for Molecule Optimization. *AAAI* (2020).
[11] Pablo Gamallo et al. 2017. A perplexity-based method for similar languages discrimination. In *4-th workshop on NLP for similar languages, varieties.*
[12] W Gao et al. 2020. Deep learning in protein modeling and design. *Patterns* (2020).
[13] Alan Gelfand. 2000. Gibbs sampling. *J. American statistical Association* (2000).
[14] Stuart Geman and Donald Geman. 1984. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *TPAMI* (1984).
[15] Walter Gilks. 2005. Markov Chain Monte Carlo. *Encyclopedia of biostat.* (2005).
[16] Bryan D He et al. 2016. Scan Order in Gibbs Sampling: Models in Which it Matters and Bounds on How Much. In *NIPS*.
[17] Weihua Hu et al. 2019. Strategies for pre-training graph neural networks. *ICLR* (2019).
[18] Kexin Huang et al. 2020. DeepPurpose: a deep learning library for drug–target interaction prediction. *Bioinformatics* (2020).
[19] Kexin Huang et al. 2021. Therapeutics data Commons: machine learning datasets and tasks for therapeutics. *NeurIPS Track Datasets and Benchmarks* (2021).
[20] John Ingraham et al. 2019. Generative Models for Graph-Based Protein Design. *NeurIPS* (2019).
[21] Wengong Jin et al. 2022. Iterative refinement graph neural network for antibody sequence-structure co-design. *ICLR* (2022).
[22] Andrew Leaver-Fay et al. 2011. ROSETTA3: an object-oriented software suite for the simulation and design of macromolecules. In *Methods in enzymology.*
[23] Chengbo Li et al. 2013. An efficient augmented Lagrangian method with applications to total variation minimization. *Computational Optimization* (2013).
[24] Ge Liu et al. 2020. Antibody complementarity determining region design using high-capacity machine learning. *Bioinformatics* (2020).
[25] Jun S Liu et al. 2001. *Monte Carlo strategies in scientific computing.* Springer.
[26] Amy X Lu et al. 2020. Self-supervised contrastive learning of protein representations by mutual information maximization. *BioRxiv* (2020).
[27] Shitong Luo et al. 2021. A 3D Generative Model for Structure-Based Drug Design. *NeurIPS* (2021).
[28] H Narayanan et al. 2021. Machine learning for biologics: opportunities for protein engineering, developability, and formulation. *Trends in pharmaco. sci.* (2021).
[29] James O'Connell et al. 2018. SPIN2: Predicting sequence profiles from protein structures using deep neural networks. *Proteins: Structure, Function, and Bioinformatics* (2018).
[30] Christine A Orengo et al. 1997. CATH–a hierarchic classification of protein domain structures. *Structure* (1997).
[31] Cristian Pasarica and Andrew Gelman. 2010. Adaptively scaling the Metropolis algorithm using expected squared jumped distance. *Statistica Sinica* (2010).
[32] Yifei Qi et al. 2020. DenseCPD: improving the accuracy of neural-network-based computational protein sequence design with DenseNet. *JCIM* (2020).
[33] Prajit Ramachandran et al. 2017. Searching for activation functions. *arXiv* (2017).
[34] Donatas Repecka et al. 2021. Expanding functional protein sequence spaces using generative adversarial networks. *Nature Machine Intelligence* (2021).
[35] Victor Garcia Satorras et al. 2021. E(n) equivariant graph neural networks. *ICML* (2021).
[36] Sam Sinai et al. 2017. Variational auto-encoding of protein sequences. *arXiv* (2017).
[37] Alexey Strokach et al. 2020. Fast and flexible protein design using deep graph neural networks. *Cell Systems* (2020).
[38] Kathryn E Tiller et al. 2015. Advances in antibody design. *Annual review of biomedical engineering* (2015).
[39] Jérôme Tubiana et al. 2019. Learning protein constitutive motifs from sequence data. *Elife* (2019).
[40] Max Welling et al. 2011. Bayesian learning via stochastic gradient Langevin dynamics. In *ICML*.
[41] Kaizhi Yue and Ken A Dill. 1992. Inverse protein folding problem: designing polymer sequences. *Proceedings of the National Academy of Sciences* (1992).
[42] Yuan Zhang et al. 2020. ProDCoNN: Protein design using a convolutional neural network. *Proteins: Structure, Function, and Bioinformatics* (2020).
[43] Jun Zhao et al. 2018. In silico methods in antibody design. *Antibodies* (2018).
[44] Yue Zhao et al. 2021. Pyhealth: A python library for health predict models. *arXiv* (2021).

A IMPLEMENTATION DETAILS

This section elaborates on the implementation details for future reproduction, including network architecture, hyperparameter setting, software/hardware configuration, etc. We divide the implementation into three components as follows.

(i) Equivariant Graph neural network (EGNN) (Section 3.2.1). For each amino acid in a sequence of training data, we mask it and use the spatial k-nearest neighbor nodes (including the masked node) as the input of EGNN, here $k = 25$. That is, when predicting the label of masked node, the nearest 25 nodes including itself are used as the input to EGNN. The label to predict is the category of the mask node. Following [35], the latent dimension is set to 100. Number of layers is set to $L = 5$. Following [35], $\phi_x(\cdot), \phi_m(\cdot), \phi_h(\cdot)$ are all two-layer MLPs. To achieve non-linearity, the Swish activation function [33] is embedded in the hidden layer. For each amino acid category, the mask node has an embedding vector, which is also learnable. During the training procedure, the batch size is set to 64, the dropout rate is set to 0.5, cross-entropy loss is used as loss criteria. We use Adam as the optimizer with an initial learning rate $1e^{-3}$. The training epoch is set to 10. The EGNN model after training 5 epoches got the highest prediction accuracy on the validation set and is used during inference.

(ii) Protein BERT (Section 3.2.2). The implementation of Protein BERT mostly follows BERT in the natural language processing community. First, it uses (1) amino acid level embedding[2]; (2) secondary structure symbols' embedding[3]; (3) positions' embedding (position is the index of the amino acid token in the whole sequence, i.e., embedding for 0,1,2,$\cdots$). The maximal length of the BERT model is set to 512. The latent dimension for all the transformers are set to 100. In each data sample, 5 amino acids are masked and predicted. Each amino acid sequence can generate 10 data samples by randomly selecting various masked amino acids. Three-layer transformers are stacked with multi-head self-attention mechanism. Tanh is used as the activation function. An MLP is attached to the representation of the last layer transformer to make the prediction. Similar to the training of EGNN, Adam is used as the optimizer with initial learning rate $1e^{-3}$.

(iii) Sampling (Section 3.3 and 3.4). $0 < \gamma < 1$ is an important hyperparameter defined in Equation (9), which determined the weight of two neural networks' predictions. We select γ from $\{0.1, 0.2, 0.3, \cdots, 0.8, 0.9\}$, evaluate their performance on validation set, and find $\gamma = 0.7$ achieves the highest recover rate and the lowest perplexity. Thus we set γ to be 0.7. λ is another critical hyperpameter when evaluating adaptive weight $\mathbf{q} = [q_1, \cdots, q_N]$ (Equation 14). In Algorithm 1, K is set to 50. η (Equation 18) is tuned on validation set and set to 0.5. Then with respect to the sampling iteration, first, in order to obtain an estimate of $D(i)$, we randomly scan over every the amino acid in the sequence twice. Then we conduct adaptive sampling as described in Section 3.3 and Algorithm 1. The sampling iteration is set to ten times of the amino acid sequence length.

Software/hardware configuration. Our method is implemented using Pytorch 1.7.0, Python 3.7, on an Intel Xeon E5-2690 machine with 256G RAM and 8 NVIDIA Pascal Titan X GPUs.

[2]20+1 categories, including masked token, as shown in Table 4

[3]8 categories, as shown in Table 4

Table 4: Symbols for all the 20 natural amino acids, 8 secondary structures and their frequencies.

20 natural amino acids			8 secondary structures		
class	symbol	%	symbol	description	%
Glycine	Gly / G	7.6%	H	Alpha helix	31.2%
Alanine	Ala / A	7.7%	B	beta-bridge	1.3%
Valine	Val / V	7.0%	E	Strand	24.1%
Leucine	Leu / L	8.6%	G	3-10 helix	3.4%
Isoleucine	Ile / I	5.5%	I	Pi helix	0.02%
Proline	Pro / P	4.6%	T	Turn	11.8%
Phenylalanine	Phe / F	3.6%	S	Bend	9.4%
Tyrosine	Tyr / Y	3.1%	-	None	18.9%
Tryptophan	Trp / W	1.2%			
Serine	Ser / S	6.7%			
Threonine	Thr / T	5.7%			
Cystine	Cys / C	1.3%			
Methionine	Met / M	2.7%			
Asparagine	Asn / N	4.1%			
Glutarnine	Gln / Q	3.9%			
Asparticacid	Asp / D	5.3%			
Glutamicacid	Glu / E	6.5%			
Lysine	Lys / K	6.3%			
Arginine	Arg / R	5.3%			
Histidine	His / H	3.1%			

Table 4 list all the amino acids (20 kinds), secondary structure (8 kinds) respectively and their frequencies. The frequencies are evaluated on all the processed protein data as described in Section 4.1.

B PROOF OF LEMMA 1

Proof. For a Markov chain, to guarantee its ergodicity, it is sufficient to prove its irreducibility and aperiodicity [15, 25]. The state here is the whole amino acid sequence. (I) **Irreducibility**: A Markov chain in which every state can be reached from every other state is called an irreducible Markov chain. If a Markov chain is not irreducible but absorbable, the sequences of microscopic forms may be trapped into some independent closed states and never escape from such undesirable states [15]. (II) **Aperiodicity**: On the other hand, a state S is aperiodic if the times of possible (positive probability) return to S have a largest common denominator equal to one [25].

(I) Irreducibility. First, we prove the irreducibility of the Markov chain. Without loss of generalization, we need to prove that any amino acid sequence pairs (Y, Z)[4] can communicate with each other, i.e., $Y \leftrightarrow Z$. Both Y and Z are two different states of the Markov chain. We prove it via constructing two possible chains: one is from Y to Z and another is from Z to Y. Both have positive probabilities. First, we want to show $Y \longrightarrow Z$, i.e., the state Y is accessible from state Z. This boils down to prove there exists an positive integer $n \in \mathbb{N}_+$ such that $P^n_{Y,Z} > 0$, here n is the step size. $P^n_{Y,Z} > 0$ denotes the probability that given the start state Y, the Markov chain takes n steps to reach the state Z. To proceed it, we construct such a Markov chain

$$\{Y^{(0)}, Y^{(1)}, \cdots, Y^{(n)}\}, \tag{22}$$

where $Y^{(0)} = Y$ and $Y^{(n)} = Z$. Specifically, we let n be the distance between Y and Z. The distance between Y and Z is defined as the total number of non-equal amino acids at all the corresponding positions, defined as $\text{Distance}(Y, Z) = \sum_{i=1}^{N} \mathbb{1}\big((Y)_i \neq (Z)_i\big)$, where Y_i and Z_i are the i-th amino acids in sequence Y and Z, respectively. The lengths of Y and Z are both equal to N. The indicator function

[4]Y and Z are conditioned on the same 3D graph structure, thus they have the same length. The length is denoted N.

$\mathbb{1}(\cdot)$ is equal to 0 when two amino acids are same and 1 otherwise. Also, we let $\mathcal{W} = \{i \mid i \in \{1, 2, \cdots, N\}$ and $(Y)_i \neq (Z)_i\}$ to collect all the indexes of amino acid where the sequence Y differs from Z. The cardinality of $\mathcal{W}$ is n, i.e., $|\mathcal{W}| = n$. Then we generate a list of all the elements in $\mathcal{W}$: $[k_1, \cdots, k_n]$, where $k_1, \cdots, k_n \in \mathcal{W}$ are n distinct indexes. For any $i = 1, \cdots, n$, we let $Y^{(i)}$ differs the previous state $Y^{(i-1)}$ on the k_i-th amino acid. The k_i-th amino acid in $Y^{(i)}$ is the same as the final state $Y^{(n)} = Z$ while the remaining amino acids are the same as the previous state $Y^{(i-1)}$. Based on acceptance rate defined in Eq. (21), we have $P^1_{Y^{(i-1)}, Y^{(i)}} > 0$ holds for any $i = 1, \cdots, n$ (the proposal distribution is a categorical distribution and output of a softmax layer, as mentioned in Section 3.2, the probability of all the elements are greater than 0). Thus, we have $P^n_{Y,Z} = P^n_{Y^{(0)}, Y^{(n)}} > \prod_{i=1}^n P^1_{Y^{(i-1)}, Y^{(i)}} > 0$, where the first ">" satisfies because we have only construct one possible Markov chain. Each permutation of $[k_1, \cdots, k_n]$ corresponds to a unique Markov chain. There are totally $n$possible chains that transforms $Y^{(0)} = Y$ to $Y^{(n)} = Z$ in n steps, each step changes an amino acid. Similarly, we can show $Z \to Y$, i.e., $\exists\ n \in \mathbb{N}$ for $P^n_{Z,Y} > 0$. We can simply reverse the Markov chain above (Equation 22) as the new Markov chain. That is, the new Markov chain is $\{Y^{(n)}, Y^{(n-1)}, \cdots, Y^{(0)}\}$, where $Y^{(n)} = Z$ and $Y^{(0)} = Y$. Now we have proved $Z \leftrightarrow Y$ hold for any amino acid sequence pairs (Y, Z). That is, we have proved the irreducibility of the Markov chain.

(II) Aperiodicity. Next, we prove the aperiodicity of the Markov chain. there is a simple test: if there is a state Y for which the 1-step transition probability $p(Y, Y) > 0$, then the chain is aperiodic [15]. There is an amino acid sequence of which the acceptance probability is lower than 1, i.e., possible to reject the proposal. The 1-step transition probability is greater than 0, so aperiodicity holds. □

C PROOF OF THEOREM 1

Proof. We consider a Markov chain on state space Ω with transition matrix M and stationary distribution π, following [16]. Let M_i be the transition matrix corresponding to sampling variable x_i. Also, let q_i ($i \in \{1, \cdots, N\}$) represent the sampling probability of the i-th variable and that $\sum_{i=1}^N q_i = 1$. For ease of exposition, the state space is augmented. The augmented state space is $\Psi = \Omega \times [n]$, including both the current state and the index of the variable to be sampled. The transition probability is $P((x, i), (y, j)) = q_j M_i(x, y)$. It can be shown that applying the transition matrix for non-uniform scan does not change the expected target distribution π, which satisfies that $\pi((x, i)) = q_i \pi(x)$. That is, $\sum_{x,i} \pi((x, i))P((x, i), (y, j)) = q_j \cdot \pi(y) = \pi(y, j)$. Thus, the stationary distribution of adaptive weight sampling is $\pi((x, i)) = q_i \pi(x)$. Proved. □

D PROOF OF THEOREM 2

Proof. In general MCMC methods, the detailed balance condition is the sufficient condition to prove the invariant distribution for the whole Markov chain. We suppose $p(\cdot)$ is target probability distribution, x and y are two states. **Detailed balanced condition** is formally defined as $p(x)\mathcal{T}(x \to y) = p(y)\mathcal{T}(y \to x)$, where $p(\cdot)$ is the target distribution for drawing samples. $\mathcal{T}(x \to y)$ is the transition kernel (a.k.a. transition probability) from the state x to the state y. **Transition probability/kernel** $\mathcal{T}(x \to y)$ is defined as the product of density of proposal distribution $q(y|x)$ and the acceptance rate $\mathcal{A}(x \to y)$, i.e., $\mathcal{T}(x \to y) = q(y|x)\mathcal{A}(x \to y)$.

Below we first show within a single step of our method, under the transition kernel defined in Equation (9) (proposal distribution) and Equation (21) (acceptance rate), the detailed balance condition holds for any neighboring sample states ($\mathrm{S}^{(t-1)}$ and $\mathrm{S}^{(t)}$), then we extend the detailed balance condition into the whole Markov chain. First, for a single step, we have

$$\begin{aligned}
&\widetilde{P}(\mathrm{S}^{(t-1)})\mathcal{T}(\mathrm{S}^{(t-1)} \to \mathrm{S}^{(t)}) = \widetilde{P}(\mathrm{S}^{(t-1)})q(\mathrm{S}^{(t)}|\mathrm{S}^{(t-1)})\mathcal{A}(\mathrm{S}^{(t-1)} \to \mathrm{S}^{(t)}) \\
=&\widetilde{P}(\mathrm{S}^{(t-1)})Q_\theta(\mathrm{s}^{(t)}_{j^{(t)}}|\mathrm{S}_{-j^{(t)}}, \mathcal{G}) \min\left\{1, \frac{\widetilde{P}(\mathrm{S}^{(t)})Q_\theta(\mathrm{s}^{(t-1)}_{j^{(t)}}|\mathrm{S}_{-j^{(t)}}, \mathcal{G})}{\widetilde{P}(\mathrm{S}^{(t-1)})Q_\theta(\mathrm{s}^{(t)}_{j^{(t)}}|\mathrm{S}_{-j^{(t)}}, \mathcal{G})}\right\} \\
=& \min\left\{\widetilde{P}(\mathrm{S}^{(t)})Q_\theta(\mathrm{s}^{(t-1)}_{j^{(t)}}|\mathrm{S}_{-j^{(t)}}, \mathcal{G}), \widetilde{P}(\mathrm{S}^{(t-1)})Q_\theta(\mathrm{s}^{(t)}_{j^{(t)}}|\mathrm{S}_{-j^{(t)}}, \mathcal{G})\right\}
\end{aligned} \tag{23}$$

Similarly, we simplify another side of detailed balance condition

$$\begin{aligned}
&\widetilde{P}(\mathrm{S}^{(t)})\mathcal{T}(\mathrm{S}^{(t)} \to \mathrm{S}^{(t-1)}) = \widetilde{P}(\mathrm{S}^{(t)})q(\mathrm{S}^{(t-1)}|\mathrm{S}^{(t)})\mathcal{A}(\mathrm{S}^{(t)} \to \mathrm{S}^{(t-1)}) \\
=& \min\left\{\widetilde{P}(\mathrm{S}^{(t-1)})Q_\theta(\mathrm{s}^{(t)}_{j^{(t-1)}}|\mathrm{S}_{-j^{(t-1)}}, \mathcal{G}), \widetilde{P}(\mathrm{S}^{(t)})Q_\theta(\mathrm{s}^{(t-1)}_{j^{(t-1)}}|\mathrm{S}_{-j^{(t-1)}}, \mathcal{G})\right\}
\end{aligned} \tag{24}$$

Combining Equation (23) and (24), we have

$$\widetilde{P}(\mathrm{S}^{(t-1)})\mathcal{T}(\mathrm{S}^{(t-1)} \to \mathrm{S}^{(t)}) = \widetilde{P}(\mathrm{S}^{(t)})\mathcal{T}(\mathrm{S}^{(t)} \to \mathrm{S}^{(t-1)}) \tag{25}$$

That is, the detailed balance condition (defined above) holds. Then we integrate out $\mathrm{S}^{(t-1)}$ on both sides of Equation (25), we obtain $\widetilde{P}(\mathrm{S}^{(t)}) = \int \widetilde{P}(\mathrm{S}^{(t)})\mathcal{T}(\mathrm{S}^{(t)} \to \mathrm{S}^{(t-1)})d\mathrm{S}^{(t-1)} = \int \widetilde{P}(\mathrm{S}^{(t-1)})\mathcal{T}(\mathrm{S}^{(t-1)} \to \mathrm{S}^{(t)})d\mathrm{S}^{(t-1)}$ We find that the transition kernel $\mathcal{T}$ would not change the distribution $\widetilde{P}$. That is, distribution $\widetilde{P}(\cdot)$ is a stationary distribution with a transition kernel $\mathcal{T}(\cdot \to \cdot)$, i.e., $\mathcal{T}(\widetilde{P}) = \widetilde{P}$ hold for transition kernel. Proved. □

E PROOF OF THEOREM 3

Proof. First, we expand the KL divergence between the target probability distribution $P(\mathrm{S}|\mathcal{G})$ and the approximate distribution $\widetilde{P}(\mathrm{S})$ as: $\mathrm{KL}(P(\mathrm{S}|\mathcal{G})\|\widetilde{P}(\mathrm{S})) = \int P(\mathrm{S}|\mathcal{G}) \ln \frac{P(\mathrm{S}|\mathcal{G})}{\widetilde{P}(\mathrm{S})} d\mathrm{S}$. Then we focus on bounding the term $\ln \frac{P(\mathrm{S}|\mathcal{G})}{\widetilde{P}(\mathrm{S})}$ and expand it as:

$$\begin{aligned}
&\left|\ln \frac{P(\mathrm{S}|\mathcal{G})}{\widetilde{P}(\mathrm{S})}\right| = \left|\ln \frac{\exp(\delta\mathcal{R}(\mathrm{S}, \mathrm{S}_{\mathrm{truth}}))}{\exp(\delta(\xi_1 - \xi_2 \mathrm{Perplexity}(\mathrm{S})))}\right| \\
=& \left|\delta\mathcal{R}(\mathrm{S}, \mathrm{S}_{\mathrm{truth}}) - \delta(\xi_1 - \xi_2 \mathrm{Perplexity}(\mathrm{S}))\right| \\
<& \delta\big(|\xi_1 - \xi_1^{\mathrm{truth}}| + |\xi_2 - \xi_2^{\mathrm{truth}}| \cdot \mathrm{Perplexity}(\mathrm{S}) + \epsilon\big) = \delta(\tau_1 + \tau_2\rho + \tau_3).
\end{aligned} \tag{26}$$

Thus, the KL divergence is bounded as

$$\begin{aligned}
&\mathrm{KL}(P(\mathrm{S}|\mathcal{G})\|\widetilde{P}(\mathrm{S})) = \int P(\mathrm{S}|\mathcal{G}) \ln \frac{P(\mathrm{S}|\mathcal{G})}{\widetilde{P}(\mathrm{S})} d\mathrm{S} \\
\leq& \int P(\mathrm{S}|\mathcal{G}) \left[\max_{\mathrm{S}} \left|\ln \frac{P(\mathrm{S}|\mathcal{G})}{\widetilde{P}(\mathrm{S})}\right|\right] d\mathrm{S} \\
=&\max_{\mathrm{S}} \left|\ln \frac{P(\mathrm{S}|\mathcal{G})}{\widetilde{P}(\mathrm{S})}\right| < \delta(\tau_1 + \tau_2\rho + \tau_3).
\end{aligned} \tag{27}$$

Proved. □