DE NOVO PROTEIN DESIGN USING GEOMETRIC VECTOR FIELD NETWORKS

Anonymous authors
Paper under double-blind review

1 VFN-IF OUTPERFORMS THE OFFICIAL PROTEINMPNN IN DESIGNABILITY.

As far as we know, VFN-IF is the first method to surpass ProteinMPNN in designability (scRMSD and scTM). We reproduced VFN-IF based on the official GitHub repository of ProteinMPNN, aligning the settings of VFN-IF with those of ProteinMPNN. Our results are groundbreaking, showing that VFN-IF outperforms ProteinMPNN in designability by up to **14%**. This implies that VFN-IF will significantly enhance the designability of widely used protein design methods. *We will release the code and weights of this model for community use*, marking a significant contribution of this work. We have incorporated this information into Table 5 of the paper. In the next version, this information will be added to Table 4.

The experimental results are presented in Table 2 below. Due to the time limit during the discussion period, the program for the setting with num sequence = 100 is still running. This is because under this setting, each sample requires time-consuming ESMFold predictions for 100 structures. We will include the results for this setting in the next version. However, *the metrics for num sequence = 8 are the official setting used by ProteinMPNN*, and the following experiments are sufficient to demonstrate the superiority of VFN-IF.

Diversity and novelty metrics are used to evaluate the diffusion model, not the performance of inverse folding. For the sake of brevity, we have omitted diversity and novelty metrics from the table. If readers are interested, these metrics are detailed in Table 4 and Table 5 in the paper.

	Setting / Metric	Noise Scale	1.0	0.5	0.1	0.1
		Number Steps	500	500	500	100
		Number Sequences	8	8	8	8
Designability	$scTM_{0.5}$	FrameDiff + ProteinMPNN	53.58%	76.42%	77.41%	76.67%
		VFN-Diff + ProteinMPNN	67.04%	81.23%	83.95%	83.83%
		VFN-Diff + VFN-IF	**72.84%**	**91.60%**	**93.46%**	**90.49%**
	$scRMSD_2$	FrameDiff + ProteinMPNN	10.62%	23.46%	28.02%	26.42%
		VFN-Diff + ProteinMPNN	25.93%	40.00%	44.20%	40.25%
		VFN-Diff + VFN-IF	**26.79%**	**53.33%**	**58.27%**	**51.36%**

Table 1: Comparison of the complete pipeline. All settings and metrics are aligned with Table 4 in the main paper. Detailed explanations for scTM and scRMSD can be found in the main paper and the appendix. Here, VFN-IF adopts the settings of ProteinMPNN.

1.1 ADDITIONAL EXPERIMENTS BASED ON RFDIFFUSION

We replaced ProteinMPNN in the RFDiffusion[1] (A concurrent study) pipeline with VFN-IF and achieved an also approximately 6% improvement in designability. The results are shown in Table :

Diffusion Models	Inverse Folding Models	$scTM_{0.5}$	$scRMSD_2$
RFDiffusion	ProteinMPNN	87%	35%
RFDiffusion	VFN-IF	**92%**	**41%**

Table 2: Comparison based on RFDiffusion.

Due to time constraints of discussion, the results above are based on the analysis of a protein with a length of 300 residues. However, this is sufficient to demonstrate the efficacy of VFN-IF.

2 THE COMPARISON BETWEEN VFN AND IPA

In this section, we first present the pseudocode for IPA and VFN to provide an intuitive comparison, as shown in subsection 2.1. Subsequently, we elaborate on the differences between IPA and VFN, as outlined in subsection 2.2. In order to further elucidate the atom representation bottleneck, we expound on how VFN addresses the bottleneck in subsection 2.3. Finally, in subsection 2.4, we introduce the pipeline of IPA to facilitate our readers' understanding of the specific processes involved in IPA.

2.1 THE PSEUDO-CODE FOR VFN AND IPA

In this section, we initially compare IPA with VFN in the provided pseudocode 1 and 2 below and explain the meaning of the notation.

Algorithm 1 The pseudo-code for the IPA module.

def IPA($\{\mathbf{s}_i\}, \{\mathbf{e}_{ij}\}, \{\mathbf{T}_i\}$):
  $\mathbf{q}_i, \mathbf{k}_i, \mathbf{v}_i = \text{Linear}(\mathbf{s}_i)$
  $\vec{\mathbf{q}}_{il}, \vec{\mathbf{k}}_{il}, \vec{\mathbf{v}}_{il} = \text{Linear}(\mathbf{s}_i)$
  $\vec{\mathbf{q}}_{il}, \vec{\mathbf{k}}_{il}, \vec{\mathbf{v}}_{il} \leftarrow \mathbf{T}_i \circ \{\vec{\mathbf{q}}_{il}, \vec{\mathbf{k}}_{il}, \vec{\mathbf{v}}_{il}\}$
  $a_{ij}^{\text{point}} = \sum_l \|\vec{\mathbf{q}}_{il} - \vec{\mathbf{k}}_{jl}\|$
  –
  $a_{ij}^{\text{node}} = \mathbf{q}_i^\top \mathbf{k}_j$
  $a_{ij}^{\text{edge}} = \text{Linear}(\mathbf{e}_{ij})$
  $a_{i,j} = \text{softmax}_j(a_{ij}^{\text{node}} + a_{ij}^{\text{edge}} + a_{ij}^{\text{point}})$
  –
  $\mathbf{o}_{ij}^{\text{node}} = \sum_j a_{i,j}\mathbf{v}_i$
  $\mathbf{o}_{ij}^{\text{edge}} = \sum_j a_{i,j}\mathbf{e}_{ij}$
  $\vec{\mathbf{o}}_{il}^{\text{point}} = \mathbf{T}_i^{-1} \circ \sum_j a_{i,j}\vec{\mathbf{v}}_{jl}$
  $\mathbf{s}_i \leftarrow \text{Linear}(\mathbf{o}_{ij}^{\text{edge}}, \mathbf{o}_{ij}^{\text{node}}, \vec{\mathbf{o}}_{il}^{\text{point}})$
  –
  return $\{\mathbf{s}_i\}$

Algorithm 2 The pseudo-code for the VFN module.

def VFN($\{\mathbf{s}_i\}, \{\mathbf{e}_{ij}\}, \{\mathbf{T}_{i\leftarrow j}\}$):
  –
  $\vec{\mathbf{q}}_{il} = \text{Linear}(\mathbf{s}_i)$
  $\vec{\mathbf{k}}_{jl} = \mathbf{T}_{i\leftarrow j} \circ \vec{\mathbf{q}}_{jl}$
  $\vec{\mathbf{h}}_k = \sum_l w_{kl}^{\text{a}} \vec{\mathbf{q}}_{il} + \sum_l w_{kl}^{\text{b}} \vec{\mathbf{k}}_{jl}$
  $\mathbf{g}_{i,j} = \text{concat}_k(\frac{\vec{\mathbf{h}}_k}{\|\vec{\mathbf{h}}_k\|}, \text{RBF}(\|\vec{\mathbf{h}}_k\|))$
  –
  –
  $a_{i,j} = \text{softmax}_j(\text{MLP}(\mathbf{s}_i, \mathbf{s}_j, \mathbf{g}_{i,j}, \mathbf{e}_{i,j}))$
  $\mathbf{v}_{i,j} = \text{MLP}(\mathbf{s}_j, \mathbf{g}_{i,j}, \mathbf{e}_{i,j})$
  $\mathbf{o}_i = \sum_j a_{i,j}\mathbf{v}_{i,j}$
  –
  –
  $\mathbf{s}_i \leftarrow \mathbf{s}_i + \text{MLP}(\mathbf{o}_i)$
  $\mathbf{e}_{i,j} \leftarrow \text{MLP}(\mathbf{s}_i, \mathbf{s}_j, \mathbf{g}_{i,j}, \mathbf{e}_{i,j})$
  return $\{\mathbf{s}_i\}, \{\mathbf{e}_{i,j}\}$

In the aforementioned pseudocode, we largely adhere to the notations used in the main text, with some distinctions in certain notations. Specifically, $\vec{\mathbf{q}}_{il}$ denotes the l-th feature vector of the i-th node, and $\vec{\mathbf{k}}_{jl}$ follows the same convention. $\mathbf{T}_i$ represents the transformation matrix from the i-th local frame (residue frame) to the global frame, while $\mathbf{T}_i^{-1}$ signifies the transformation matrix from the global frame to the i-th local frame. $\mathbf{T}_{i\leftarrow j}$ denotes the transformation matrix from the j-th local frame to the i-th local frame. $\mathbf{s}_i$ represents the representation of the i-th node, and $\mathbf{e}_{ij}$ represents the representation of the edge between i-th and j-th nodes. w_{kl}^{a} and w_{kl}^{b} are learnable weights. The specific process of VFN is detailed in the main text; the procedure for IPA is elaborated in subsection 2.4. For the sake of conciseness in comparison, the pseudocode overlooks certain non-essential factors. For instance, in the pseudocode, we illustrate a single-head attention mechanism, but in fact, both IPA and VFN employ a multi-head attention mechanism.

2.2 THE DIFFERENCES BETWEEN IPA AND VFN

As indicated in the pseudocode, the attention mechanism for virtual atoms in IPA and VFN is fundamentally different. The most crucial distinction lies in the fact that, due to the constraints of SE(3) invariance, IPA cannot directly employ an activation function when extracting features for virtual atoms, as explained in subsection 2.3. In contrast, VFN circumvents this limitation and utilizes a vector field operator to extract features, denoted as $\vec{\mathbf{h}}_k$, and complementing it with an MLP (ReLU inside) for feature extraction. To accommodate this design choice, the overall architecture of VFN diverges significantly from that of IPA.

2.3 BYPASSING IPA BOTTLENECKS

The design of IPA has led to the atom representation bottleneck, while the design of VFN avoids this issue. Specifically, in IPA, $\mathbf{q}_i$ and $\mathbf{k}_j$ are with respect to the global frame, and the operator corresponding to $\mathbf{q}_i, \mathbf{k}_j$ needs to maintain SE(3) invariance. This constraint results in the inability to directly apply activation functions on $\mathbf{q}_i, \mathbf{k}_j$, as doing so would compromise the SE(3) invariance. Consequently, IPA can only employ operations similar to distance pooling. We refer to this limitation as the atom representation bottleneck.

In VFN, we place these virtual atoms in the same local frame $\mathbf{T}_i$, utilizing the local frame to ensure SE(3) invariance, as proved in subsection A.2.6. This eliminates the need to impose constraints on operators to achieve SE(3) invariance. Operators in VFN can freely utilize activation functions without disrupting SE(3) invariance. This characteristic allows VFN to circumvent the atom representation bottleneck present in IPA.

2.4 THE PIPELINE OF IPA

As shown in pseudocode 1, IPA employs an attention mechanism. Unlike VFN, its attention mechanism consists of three parts: node attention, edge attention, and point attention. Specifically, node attention and edge attention utilize common methods. In particular, node attention employs a mechanism similar to that of the transformer, obtaining attention weights a_{ij}^{node} through dot product of $\mathbf{q}_i$ and $\mathbf{k}_j$. In edge attention, the representation of edges $\mathbf{e}_{ij}$ introduces attention bias a_{ij}^{edge} through a linear layer. Point attention is the core of IPA, where attention weights $\vec{\mathbf{o}}_{il}^{\text{point}}$ are obtained through a distance pooling operation on the virtual atomic distances ($\|\vec{\mathbf{q}}_{il} - \vec{\mathbf{k}}_{jl}\|$). Subsequently, these attention weights are summed and normalized through softmax to obtain the final attention weights $a_{i,j}$. These attention weights $a_{i,j}$ are then used to calculate weighted averages for the corresponding representations $\mathbf{v}_i, \mathbf{e}_{ij}, \vec{\mathbf{v}}_{jl}$, producing the output for each type of attention. $\mathbf{o}_{ij}^{\text{node}}, \mathbf{o}_{ij}^{\text{edge}}, \vec{\mathbf{o}}_{il}^{\text{point}}$. Finally, these outputs are collectively updated for each node's representation $\mathbf{s}_i$ through a linear layer, achieving the frame modeling.