DIFFEOMORPHIC OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Generative models learn data distributions that reside on a low-dimensional manifold within a higher-dimensional ambient space. Optimizing differentiable objectives on such a manifold is challenging: the ambient loss landscape is high-dimensional, rugged, and non-convex. Direct gradient descent, blind to the manifold's geometry, quickly drifts off it. Our approach starts from the observation that diffusion and flow models provide a map from the data manifold to a much simpler base space in which we perform gradient descent instead. Using concepts from differential geometry, we show this is equivalent to Riemannian gradient descent on the data manifold, keeping trajectories on-manifold by construction and yielding a smoother optimization surface. We extend diffeomorphic optimization to matrix groups, such as SO(3), SE(3) and SU(N) with autograd-compatible Lie-group ODE solvers, which allows us to empirically demonstrate the effectiveness of our approach in the highly relevant task of protein design and lattice gauge theory.

1 INTRODUCTION

Machine learning data is typically concentrated on a low-dimensional manifold (Fefferman et al., 2016; Brown et al., 2022; Kiani et al., 2024). In practice, these manifolds are not known explicitly. Many objectives we care about, such as the stability of a protein or the fidelity of a generated image, are in fact defined on low-dimensional data manifolds embedded in high-dimensional spaces. Direct optimization in this space is difficult: the objective landscape is highly non-convex, the data manifold is implicit, and unconstrained updates, with e.g. direct gradient descent in the data space, tend to produce out-of-distribution solutions.

Diffusion and flow-based generative models provide a means to address this difficulty. These models learn a diffeomorphic map from a tractable base distribution to the data distribution. This transformation reparameterizes the data manifold in terms of the variables of a simple base distribution. The geometry of the manifolds required for optimization are already implicitly learned by these models. While generative models have been highly successful in sampling from such manifolds, their use as a foundation for subsequent optimization is less developed.

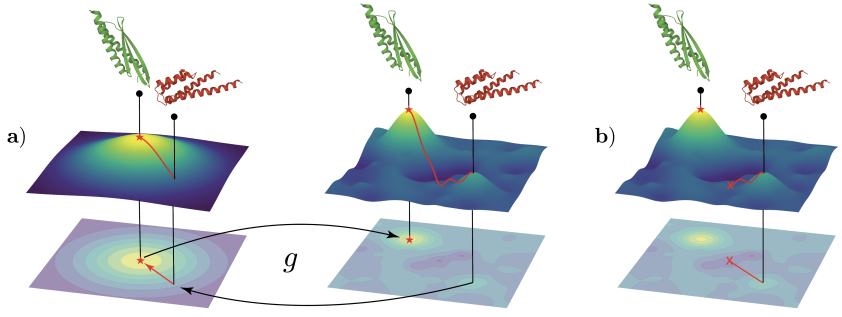


Figure 1: Optimization of protein with undesired properties (red) to a protein of desired properties (green). **Right:** Direct optimization in target space gets stuck in local minima. **Left:** Optimization in base space moves between modes while staying close to the data.

In this work, we propose diffeomorphic optimization - a novel method to optimize arbitrary differentiable cost functions on the data manifold. Our method harnesses the recent advances in flow

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

matching and diffusion models, is applicable to a wide range of tasks, and scales to high dimensional data, such as proteins with several hundreds of residues. The proposed framework yields a inference time compute method for frozen, pretrained diffusion and flow models. For a visual intuition of this idea, we refer to Figure 1.

Existing methods that try to optimize objectives with pretrained models come with caveats: guidance approximates an intractable conditional density Dhariwal & Nichol (2021); reward fine tuning requires retraining Clark et al. (2023), and optimal control methods inject control into the flow and can pull trajectories off manifold Wang et al. (2024). None of these apply to diffeomorphic optimization.

The approach diffeomorphic optimization takes is the following. Diffusion and flow models learn a diffeomorphic (smooth and invertible) map $g : \mathcal{Z} \to \mathcal{X}$ by integrating the ordinary differential equation $g(z) \equiv x_1 = z + \int_0^1 d\tau\, v_\theta(x_\tau, \tau)$ that maps a base space $\mathcal{Z}$, equipped with a simple probability density $q_\mathcal{Z}$, to the target space $\mathcal{X}$, for example, the space of atom positions. The invertible map g is parameterized by a neural network and induces a density $q_\mathcal{X}(x) = q_\mathcal{Z}(g^{-1}(x))|\det \frac{dg^{-1}}{dx}|$ on the target space $\mathcal{X}$. The training objective then aims to ensure that this pushforward density $q_\mathcal{X}(x)$ agrees with the data distribution to a good approximation. As a result, the image $\mathcal{D} := g(\mathcal{Z}) \subset \mathcal{X}$ serves as a learned model of the data manifold, and g maps base space elements $z \in \mathcal{Z}$ into the region where the data distribution concentrates with high probability. Consider a cost function $\mathcal{L} : \mathcal{X} \to \mathbb{R}$, e.g. the energy or thermostability of a protein, that we want to optimize on the learned data manifold $\mathcal{D}$. Since the map g is diffeomorphic, we change coordinates to the base space variables,

$$\mathcal{L} \circ g : \mathcal{Z} \to \mathbb{R}, \quad z \mapsto \mathcal{L}(g(z))\,. \tag{1}$$

We can then perform gradient descent with learning rate $\lambda \in \mathbb{R}$ in the base space variables

$$z^{(i+1)} = z^{(i)} - \lambda \nabla_z \mathcal{L}(g(z^{(i)}))\,, \qquad \text{for } i \in \{1, \dots, n\}\,, \tag{2}$$

and map the final base point $z^{(n)}$ to its corresponding $x^{(n)} = g(z^{(n)})$. This parameterization has several advantages: the map g is bijective and thus no information is lost. The data distribution is often highly multimodal. However, the density of the base space $q_\mathcal{Z}$ is chosen to be a simple unimodal distribution, such as a normal distribution $\mathcal{N}(0, 1)$. Thus, the base space reparameterization makes it easier to switch modes leading to a smoother loss landscape. Furthermore, since a well-trained g maps base space samples with high probability to the data manifold, the gradient-descent optimization is equivalent to gradient descent on the data manifold, as we show in Section 3 using differential geometry.

The optimization task is complicated by the fact that generative models for proteins often do not sample the atom positions directly but rather the $SE(3)$ backbone frames and $SO(2)$ sidechain dihedral angles Yim et al. (2023b;a); Jing et al. (2024); Jumper et al. (2021). We will therefore derive two efficient methods to facilitate backpropagation through ODE solvers on matrix Lie groups in Section 4. The first relies on repurposing existing autograd engines for calculating the Riemannian gradient. For the second, we derive a suitable generalization of the adjoint-state method for matrix Lie groups.

We showcase diffeomorphic optimization in several numerical experiments. Specifically, we demonstrate that we can start from a given protein and then controllably change (parts of) its secondary structures using the flow matching model FrameFlow Yim et al. (2023b) as the diffeomorphic map. Using the DiffDock diffusion model Corso et al. (2023), we show that we can optimize the Vina docking score Trott & Olson (2010) of a ligand in a protein pocket, and improve the stability and affinity of peptides Wang et al. (2024). Furthermore, we demonstrate that we can minimize the Rosetta energy function of a given protein in the base space of the AlphaFlow model Jing et al. (2024). This relaxation protocol leads to significantly lower energies than the state-of-the-art Rosetta Relax.

Our work contributes to a recent rise of inference computation for protein design. Practitioners often sample thousands of designs, rank, and submit only a handful for further wetlab experiments Bennett et al. (2024); Abramson et al. (2024); Watson et al. (2023); Frey et al. (2025). The reason for this is that experimental wet lab capacity (and not sampling costs) is the main bottleneck. Furthermore, extensive sampling has been shown to lead to pronounced performance improvements, for example, for antibody-antigen coupling Abramson et al. (2024). Diffeomorphic optimization provides a more targeted approach to obtain high quality samples compared to brute-force sampling followed by ranking and selection. It does not require finetuning (unlike RL or reward finetuning), auxiliary networks (as in guidance) or rejection sampling (best-of-N). It can be applied to any frozen, pretrained,

flow or diffusion model with a differentiable reward. We therefore believe that it will become an important part of the inference compute toolbox for proteins.

Briefly summarized, our key contributions are the following. We propose diffeomorphic optimization that is applicable to any differentiable optimization objective. As we show theoretically, it keeps optimization trajectories on the learned data manifold in regions with high probability and leads to a smoother optimization landscape. To enable this, we propose two effective methods to facilitate backpropagation through matrix Lie group ODE solvers and will provide efficient PyTorch implementations for it. Experimentally, we apply diffeomorphic optimization to tasks of high practical relevance, i.e. protein ligand docking, Rosetta energy relaxation, and secondary structure modification, by combining it with state-of-the-art generative models in the field of protein generation, specifically FrameFlow, DiffDock, and AlphaFlow.

2 Related Works

Protein hallucination is a version of computational protein design which uses backpropagation through a folding model to its input sequence Anishchenko et al. (2021); Kosugi & Ohue (2022); Goverde et al. (2023); Pacesa et al. (2024); Cho et al. (2025), and gradient descent through normalizing flows has been explored in the explainability literature to generate counterfactual explanations Joshi et al. (2019); Dombrowski et al. (2021; 2023); Dhurandhar et al. (2018), although not for flow matching and diffusion models. Ben-Hamu et al. (2024) proposes to differentiate through flows for controlled generation. Our work builds on this reference by generalizing it to matrix Lie groups, which is of high relevance for proteins, and by deriving an efficient adjoint state method. Wang et al. (2024) and Liu et al. (2023) instead add a control to the vector field of the flow, whereas our approach optimizes the initial condition following Ben-Hamu et al. (2024); we compare to Wang et al. (2024) in detailed numerical experiments. The adjoint state method on manifolds has been discussed in other works for charts Lou et al. (2020); Mathieu & Nickel (2020) or particular manifolds Rezende & Mohamed (2020); Bacchio et al. (2023); Albergo & Vanden-Eijnden (2022). A widely used alternative for biasing diffusion and flow-matching models towards certain desiderata is guidance, which comes in various flavors such as classifier-based Dhariwal & Nichol (2021), classifier-free Ho & Salimans (2022) and universal guidance Bansal et al. (2023). An expanded discussion of all of these references is given in Appendix B.

3 Diffeomorphic Optimization stays on Manifold

In this section, we analyze the diffeomorphic optimization procedure theoretically using differential geometry. In particular, we will demonstrate its relation with gradient descent on the manifold.

We briefly define the notions that we use and refer to Appendix D and the textbook Lee (2018) for a self-contained review. A manifold $\mathcal{M}$ is a space that locally takes the form of $\mathbb{R}^D$, with coordinates x^i supplied by charts. The tangent space $T_p\mathcal{M}$ at a point $p \in \mathcal{M}$ collects the velocities of curves through p, and a smooth map $g : \mathcal{M} \to \mathcal{V}$ acts on it by its differential $dg_p : T_p\mathcal{M} \to T_{g(p)}\mathcal{V}$. A Riemannian metric G equips every tangent space with an inner product $\langle \cdot, \cdot \rangle_{G_p}$ and thereby defines the Riemannian gradient $\operatorname{grad}_p^G f \in T_p\mathcal{M}$ of a function f as the unique tangent vector with $\langle v, \operatorname{grad}_p^G f \rangle_{G_p} = df_p[v]$ for all $v \in T_p\mathcal{M}$, which in coordinates reads $\widehat{\operatorname{grad}_p^G f} = \sum_j G^{ij}\partial_j f$ with G^{ij} the inverse metric tensor. The metric furthermore defines geodesics and the exponential map $\exp_p : T_p\mathcal{M} \to \mathcal{M}$, which takes a step along a tangent vector while remaining on the manifold. Gradient descent on a Riemannian manifold is then given by $p^{i+1} = \exp_{p^i}(-\lambda \operatorname{grad}_{p^i}^G \mathcal{L})$ with learning rate λ, which reduces to standard gradient descent in $\mathbb{R}^n$ where $\exp_p(v) = p + v$.

3.1 Diffeomorphic optimization is equivalent to Gradient Descent on the Data Manifold

Let $g : \mathcal{Z} \to \mathcal{X}$ be a generative model that maps its latent space $\mathcal{Z}$ to the target space $\mathcal{X}$, and recall that $\mathcal{D} = g(\mathcal{Z})$ denotes its image, carrying the pushforward density $q_{\mathcal{X}}$. Given such a model, we can then perform gradient descent in its latent space and map the resulting point to $\mathcal{D}$. We then show:

Theorem 1. *Let $g : \mathcal{Z} \to \mathcal{D}$ be a diffeomorphic generative model onto its image $\mathcal{D} = g(\mathcal{Z})$ and let $\mathcal{L} : \mathcal{D} \to \mathbb{R}$ be the loss. Then, up to quadratic corrections in the learning rate λ, performing gradient descent in the latent space $\mathcal{Z}$ and then mapping it to $\mathcal{D}$ with g is equivalent to Riemannian gradient descent on $\mathcal{D}$, i.e.,*

$$g(\exp_z(-\lambda\, \mathit{grad}^{\tilde{G}}_z \mathcal{L} \circ g)) = \exp_{g(z)}(-\lambda\, \mathit{grad}^{G}_{g(z)} \mathcal{L} + \mathcal{O}(\lambda^2))\,, \tag{3}$$

where $G(u, v) = \tilde{G}(dg^{-1}u, dg^{-1}v)$ denotes the pushforward of the Riemannian metric $\tilde{G}$ on $\mathcal{Z}$.

Proof: see Appendix C.1. In coordinates, the gradient in data space is given by

$$\widehat{\text{grad}}_{g(\hat{z})} \mathcal{L} = J_g\, \widehat{\text{grad}}_z \mathcal{L}\,, \qquad \text{with } J_g = \frac{\partial \hat{g}}{\partial \hat{z}}\,. \tag{4}$$

We can perform a SVD decomposition of the Jacobian J_g. In the basis of its left singular values, the learning rate will be scaled by the corresponding singular vectors. The model density on $\mathcal{D}$ is heavily concentrated around the data manifold and thus we expect low values for the singular values along these contracted directions. This gives us a mechanism to extract the implicitly learned tangent space from the generative diffusion or flow-matching model, which is (approximately) spanned by the remaining left-singular vectors.

4 Backpropagation through $\mathbf{SE(3)}$ ODE solvers

The frame representation of proteins considers an idealized backbone geometry of its heavy atoms $[N, C_\alpha, C, O] \in \mathbb{R}^{3,4}$ determined by fixed bond lengths and angles Jumper et al. (2021); Yim et al. (2023b). A protein consists of multiple residues and for each residue i, the main backbone atoms can be described by a simple rotation $R_i \in SO(3)$ and translation $t_i \in \mathbb{R}^3$ of these idealized backbone coordinates, as seen in Figure 2,

$$[N_i, C_i, (C_\alpha)_i] = T_i[N, C, C_\alpha] \tag{5}$$

where $T_i = (R_i, t_i)$ is an element of the three-dimensional special Euclidean group $SE(3) = SO(3) \ltimes \mathbb{R}^3$. The positions of the remaining heavy atoms (backbone oxygen and sidechain carbons) can be fixed by dihedral angles. Therefore, the vector field v_θ in the ODE takes value in the tangent space of the corresponding Lie groups $SE(3)$ and $SO(2)$. Integration on $SO(2)$ can simply be performed by standard numerical integration in $\mathbb{R}^3$ and then wrapping the result on the circle S^1. The case of $SE(3)$ is, however, highly non-trivial. For example, the group $SE(3)$ has the product rule

$$T_1 T_2 = (R_1, t_1)(R_2, t_2) = (R_1 R_2, R_1 t_2 + t_1)\,. \tag{6}$$

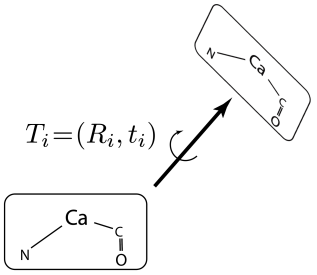


Figure 2: $SE(3)$ roto-translation of the idealized backbone positions.

The translation group $\mathbb{R}^3 \simeq \{(I, t) | t \in \mathbb{R}^3\}$, where I is the unit element of $SO(3)$, is an abelian normal subgroup of $SE(3)$ since $(R, t)(I, t')(R, t)^{-1} = (I, Rt')$. Therefore $SE(3)$ is not semi-simple and there is no canonical left- and right-invariant Riemannian metric induced by the Killing form. In the generative model literature, one typically chooses the Riemannian metric to be the sum of the metrics of $SO(3)$ and $\mathbb{R}^3$ Yim et al. (2023b). This choice is right-invariant but not left-invariant by our convention (Appendix D). For this choice, the exponential maps and Riemannian gradients for the translations and rotations decouple

$$\exp_T((\omega, v) = (\exp_R(\omega), \exp_t(v))\,, \qquad \text{grad}_T f = (\text{grad}_R f, \text{grad}_t f)\,, \tag{7}$$

for $\omega \in \mathfrak{so}(3)$, $v \in \mathfrak{r}^3 \simeq \mathbb{R}^3$, and $f : SE(3) \to \mathbb{R}$. The explicit form of these exponential maps and gradients for both translations and rotations are:

Translations: since $\mathbb{R}^3$ is a vector space, it holds that

$$\exp_t(v) = t + v\,, \qquad \text{grad}_t f = \nabla_t f(t, R) \tag{8}$$

where we use the canonical isomorphism $\mathfrak{r}^3 \simeq \mathbb{R}^3$ and ∇_t denotes the standard gradient operator $\nabla = (\partial_1, \partial_2, \partial_3)$. As a result, we recover standard gradient descent for the translational part

$$t^{i+1} = t^{(i)} - \lambda \nabla_{t^i} \mathcal{L}(t^i, R^i)\,. \tag{9}$$

Rotations: as we discuss in Appendix D, the right multiplication $R_g : G \to G$, $h \mapsto hg$ induces a isomorphism between the tangent spaces T_gG and the Lie algebra $\mathfrak{g} \simeq T_eG$ for any Lie group G. It is often easier to work with the exponential map $\exp : \mathfrak{g} \to G$ in terms of Lie algebra elements. For matrix Lie groups, such as $SO(3)$, this exponential map is simply given in terms of the matrix exponential $\exp(v) = \sum_{i=0}^{\infty} \frac{v^n}{n!}$ where v is the matrix Lie algebra element. Similarly, we can uniquely associate the Riemannian gradient $\text{grad}_R f$ of a function $f : SO(3) \to \mathbb{R}$ with the Lie algebra element

$$\nabla f(R) = \sum_a T^a \nabla^a f \in \mathfrak{so}(3)\,, \qquad \nabla^a f(R) = \frac{d}{dt} f(\exp(tT^a)R)\Big|_{t=0}\,, \tag{10}$$

where T^a denote the antisymmetric generators of the Lie algebra $\mathfrak{so}(3)$. Therefore, gradient descent on $SO(3)$ amounts to

$$R^{i+1} = \exp(-\lambda \nabla_{R^i} \mathcal{L}(R^i, t^i)) R^i\,. \tag{11}$$

Summary: Gradient descent on $SE(3)$ can be performed by

$$T^{i+1} = \begin{bmatrix} R^{i+1} \\ t^{i+1} \end{bmatrix} = \exp_{T^i}(-\lambda \nabla_{T^i} \mathcal{L}(T^i)) = \begin{bmatrix} \exp\left(-\lambda \nabla_{R^i} \mathcal{L}(R^i, t^i)\right) R^i \\ t^{(i)} - \lambda \nabla_{t^i} \mathcal{L}(t^i, R^i) \end{bmatrix} \tag{12}$$

In this sense, the gradient descent of the rotational and translational part decouples.

4.1 Gradients of $\mathbf{SE(3)}$ Solvers

Let $g : SE(3)^n \to SE(3)^n$ be a diffeomorphism. We restrict to $n = 1$ for notational simplicity but the results immediately generalize to $n > 1$. We use the notation $(R, t) = g(Z, z)$ with $Z \in SO(3)$ and $z \in \mathbb{R}^3$ denoting the base space variables. We can thus reparameterize the loss function $\mathcal{L} : SE(3) \to \mathbb{R}$ by

$$\mathcal{L}(g(Z, z)) \tag{13}$$

for which we want to perform gradient descent with respect to the base variables $(Z, z) \in SE(3)$. In particular, we can consider flow matching and diffusion models for which the diffeomorphism g is defined by $g(Z, z) \equiv (Z_1, z_1)$ where the right hand side is the solution of the following initial value problem on $SE(3)$:

$$dT_\tau = (dZ_\tau, dz_\tau) = (\, V_\theta(Z_\tau, z_\tau),\, v_\theta(Z_\tau, z_\tau)\,)\, d\tau\,, \quad \text{with} \quad T_0 = (Z_0, z_0) = (Z, z)\,, \tag{14}$$

where $\tau \in [0, 1]$ and $V_\theta \in \mathfrak{so}(3)$, $v_\theta \in \mathbb{R}^3$ are parameterized by neural networks. As a result, we need to backpropagate through the numerical solver of the ODE. Specifically, we need to calculate the Riemannian gradient $\nabla_Z \mathcal{L}(g(Z, z))$ with respect to rotation $Z \in SO(3)$. In the following, we will propose two methods to do so.

Repurposing Autograd for $\mathbf{SO(3)}$: the Riemannian gradient can be expressed in terms of a simple matrix derivative:

Theorem 2. *The Riemannian gradient equation 10 on $SO(3)$ of a loss function $\mathcal{L} : SO(3) \to \mathbb{R}$ can be written as*

$$\nabla \mathcal{L}(R) = 2 \left[\frac{df}{dR} R^\top \right]_A\,, \tag{15}$$

where $\frac{df}{dR}$ denotes the standard matrix-calculus gradient with respect to the matrix $R \in SO(3)$. Furthermore, we denote the antisymmetric component of a matrix M by $[M]_A = \frac{1}{2}(M - M^\top)$.

Proof: see Appendix C.2.

This suggests a straightforward way to calculate Riemannian gradients with modern autograd frameworks. Specifically, one wraps any $SO(3)$-valued leaf variable in an autograd method acting as the identity in the forward pass but performing the multiplication equation 15 in its backward where $\frac{df}{dR}$ is the output gradient. The gradient of this variable will correspond to the antisymmetric Riemannian gradient matrix equation 10, see Appendix A for a very concise pytorch implementation of this.

We emphasize that this approach is completely general and ensures that the Riemannian gradient is seamlessly integrated in existing autograd functionality.

In particular, we can facilitate backpropagation through ODE solvers on $SO(3)$ by combining this repurposing trick with standard activation checkpointing at intermediate points of the integration trajectory. This allows us to limit the memory footprint of the computational graph to the desired degree.

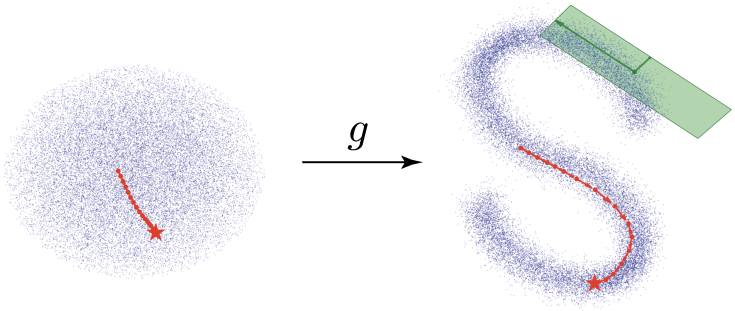


Figure 3: Diffeomorphic Optimization on $SO(3)$: left hand side visualizes the gradient descent trajectory in the base space $\mathcal{Z}$. Right hand side visualizes the same trajectory when mapped to the target space $\mathcal{X}$. Diffeomorphic optimization clearly stays on manifold. The green plane is spanned by the left-singular vectors of the Jacobian $\nabla_Z g(Z)$ scaled by the corresponding singular vectors. This shows that the diffeomorphic map indeed captures the tangent space of the data manifold to good approximation.

Adjoint State Method on $\mathbf{SE(3)}$: the repurposing method has the advantage that it is rather general. However, for flow matching and diffusion models, the gradients can be written in terms of a particular adjoint state ODE. For the translational part, this is well known Chen et al. (2018) and given by

$$\frac{d\mathcal{L}(g(Z,z))}{dz} = a_0 \tag{16}$$

where the adjoint state $a_0 \in \mathbb{R}^3$ is obtained by solving the following terminal value problem

$$\frac{da_\tau}{d\tau} = -\sum_{i=1}^{3} a_\tau^i \frac{dv_\theta^i(Z_\tau, z_\tau)}{dz_\tau}\,, \qquad\qquad a_1 = \frac{d\mathcal{L}(R,t)}{dt}\,, \tag{17}$$

with $\tau \in [0,1]$. In the appendix, we derive a generalization of the adjoint state method for $SO(3)$:

Theorem 3. *The Riemannian gradient of the reparameterized loss function*

$$\nabla_Z \mathcal{L}(g(Z,z)) = A_0 \tag{18}$$

can be obtained by the following terminal value problem for the Lie-algebra-valued adjoint state $A_\tau = \sum_{i=1}^3 A_\tau^i T^i \in \mathfrak{so}(3)$*:*

$$\frac{dA_\tau}{d\tau} = [V_\theta(Z_\tau, t_\tau), A_\tau] - \sum_{i=1}^{3} A_\tau^i \nabla_{Z_\tau} V_\theta^i(Z_\tau, t_\tau)\,, \qquad\qquad A_1 = \nabla_R \mathcal{L}(R,t)\,, \tag{19}$$

where $\tau \in [0,1]$ *and* $[A,B] = AB - BA$ *is the commutator of matrices* A *and* B*.*

Proof: see Appendix C.3; for the cross terms of the coupled flow equation 14, see Appendix C.4.

Wang et al. (2024) used the adjoint method on $SO(3)$ to calculate gradients with respect to deterministic optimal control which they used as guidance. As shown in the appendix, this result also holds for any compact matrix Lie group, where the projection onto $\mathfrak{so}(3)$ is replaced by the projection onto the respective Lie algebra. We use this for our lattice gauge theory experiments on $SU(3)$, for which the projection maps onto the traceless anti-Hermitian matrices. Like $SO(3)$, the group $SU(3)$ is compact and semi-simple, so that the bi-invariant metric induced by the Killing form in Appendix D is available. A notable advantage of the adjoint state method is that it turns the gradient calculation

into a terminal value problem and thus can benefit from modern ODE solvers. However, it assumes that the forward pass has no discretization errors, so that stiff ODEs may accumulate a systematic error. In practice, we recommend benchmarking both methods numerically for the problem at hand. We will provide a modular integration library that implements both methods.

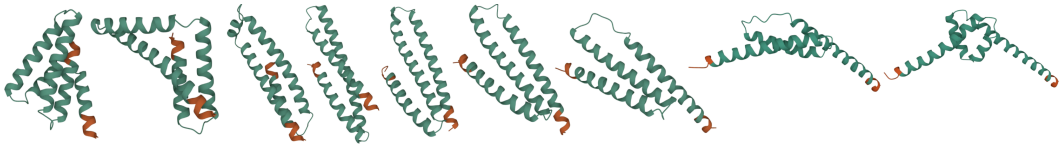
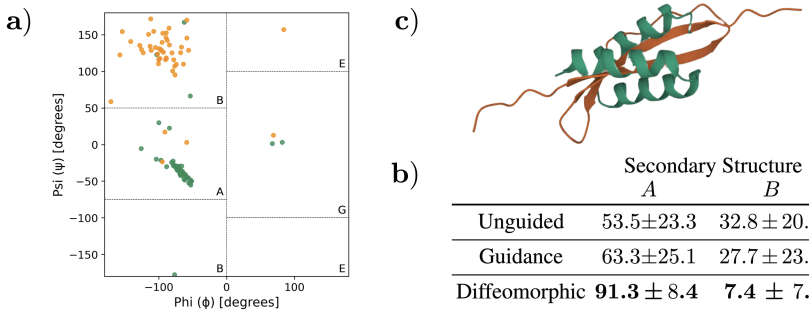


Figure 4: We show various snapshots of the trajectory of diffeomorphic maximization of the distance between the termini of the proteins (marked in red). The optimization traverses a number of plausible conformations.

5 Experiments

We demonstrate the generality of diffeomorphic optimization in several numerical experiments. We refer to Appendix H for more details.

$\mathbf{SO(3)}$ Manifold: In Figure 3, we visualize diffeomorphic optimization for a toy dataset on $SO(3)$. Specifically, we create a dataset of S-shape in the angle-axis vectors of the rotations. We then train a flow matching model to sample $SO(3)$ elements on this manifold. The diffeomorphic optimization aims to push the sample as close as possible to the point marked by a star on the S-shaped data manifold. As the figure demonstrates, diffeomorphic optimization leads to a gradient descent trajectory on the data manifold. This illustrates that diffeomorphic optimization can harness the differential geometric notions learned by the flow model.



	Secondary Structure A	B
Unguided	53.5 ± 23.3	32.8 ± 20.1
Guidance	63.3 ± 25.1	27.7 ± 23.3
Diffeomorphic	$\mathbf{91.3 \pm 8.4}$	$\mathbf{7.4 \pm 7.3}$

Figure 5: Diffeomorphic optimization is used to change the secondary structure of the protein. For this, we use the ABEGO classification shown on the left. We start from the brown protein shown on the top right whose residues are mostly clustered in the B region of the Ramachandran plot which corresponds to β-sheets. The energy is biased towards the A region, so that these residues are pushed from B into A, which corresponds to α-helices. The resulting protein structure is shown in green in the top right. The initial and final Ramachandran plots are shown on the left. The lower right shows the comparison to guidance for 50 samples obtained from the model, measured as the percentage of residues classified as A. Diffeomorphic optimization outperforms guidance despite careful tuning of the baseline described in the appendix.

Peptide Design: we compare diffeomorphic optimization to the OC-Flow framework Wang et al. (2024), which adds a control to the vector field of the flow. We kept the experimental setup identical to Wang et al. (2024) and compare to our proposed diffeomorphic optimization of both translation and rotations of the peptide's backbone. We use the repurposing approach and summarize the results in Table 1. We stress that this improvement can be obtained while being $2\times$ faster in terms of runtime.

Secondary Structure Modification with FrameFlow: we use FrameFlow Yim et al. (2023a), a widely used backbone generation flow matching model, to optimize the backbone structure with respect to a given cost function. Figure 4 illustrates an optimization trajectory for which the distance

Table 1: Evaluation of OC-Flow peptide design.

	MadraX ↓	RMSD ↓	SSR % ↑	BSR % ↑	Stability ↓	Affinity ↓	Diversity ↑
Ground-truth	-0.588	–	–	–	-84.893	-36.063	–
PepFlow	-0.195	1.645	0.794	0.874	-45.660	-26.538	0.310
OC-Flow(trans)	-0.229	1.774	0.797	0.876	-48.380	-27.328	0.323
OC-Flow(rot)	-0.221	1.643	0.794	0.872	-48.636	-27.211	0.310
OC-Flow(trans+rot)	-0.263	2.127	**0.797**	0.869	-48.853	-27.468	0.338
DiffeoOpt	**-0.309**	**1.605**	0.796	**0.881**	**-49.417**	**-28.409**	**0.340**

between the protein termini is maximized. This demonstrates that our method can optimize a given protein structure while staying on the data manifold. Similarly, Figure 5 shows that we can also optimize the secondary structure of a protein as specified by the ABEGO classification scheme Wintjens et al. (1996); Kim et al. (2009).

For this, we minimize an energy function that is based on the Ramachandran-PAAPP-combined term in the Rosetta energy Alford et al. (2017); Leaver-Fay et al. (2025) which interpolates energies derived from statistics of backbone dihedral preferences in the PDB on a toroidal grid. The grid values are modified such that the regions which do not correspond to the desired secondary structure are disfavored, which biases the energy towards whichever region we choose as the target. We use this for both the A region (α-helices) and the B region (β-sheets). We compare to guidance with carefully tuned hyperparameters. Diffeomorphic optimization outperforms guidance while maintaining designability with results in Appendix H.2.

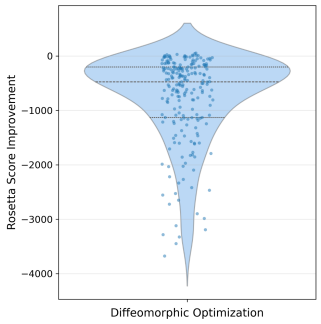


Figure 6: Diffeomorphic optimization applied to AlphaFlow exhibits strong improvement as measured in the Rosetta energy score.

Protein-Ligand Docking with DiffDock: DiffDock is a diffusion model that samples both the center of mass translation and rotation (i.e. an $SE(3)$ element) and the torsion angles of the ligand for a given protein Corso et al. (2023). The model is trained on the PDBBind dataset Wang et al. (2005), which contains experimentally determined protein-ligand docking structures. We then apply diffeomorphic optimization to maximize the popular VinaSF score of the protein-ligand complex, utilizing its OpenDock implementation Hu et al. (2024).

To enable backpropagation through the solver, we modify the DiffDock generation process to use probability ODE sampling through our custom integration library. We then optimize the score function on pdb test set samples. We compare diffeomorphic optimization to iid sampling followed by selecting the sample with the best docking score using the same computational budget. For this, each gradient descent step is counted as three sampling trajectories, which is quite generous for the baseline. Figure 7 shows the improvement $S_{\text{diffeo}} - S_{\text{iid}}$ obtained by diffeomorphic optimization which significantly outperforms this baseline. Our method can thus efficiently combine physics-based docking scores with trained generative models. Further analysis is provided in Appendix H.3.

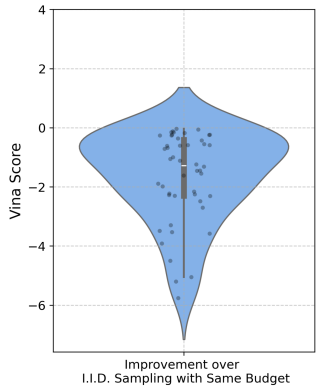


Figure 7: Diffeomorphic optimization leads to better/lower vina score than iid sampling with the same budget.

Minimization of Rosetta Energy with AlphaFlow: we consider the Rosetta energy function Alford et al. (2017) which is widely used in the protein community using the tmol pytorch implementation Leaver-Fay et al. (2025) of beta_nov2016_cart. We select the same pdb test set as in the AlphaFlow publication Jing et al. (2024).

This reference proposed several flows that are obtained from protein structure prediction models. We select the ESMFold model for convenience, as it does not require MSA processing. We compare diffeomorphic optimization to the state-of-the-art Rosetta Relax protocol. From our theoretical analysis, we expect that diffeomorphic optimization will lead to better mode mixing. Once an energetically favorable mode is reached, it is cheaper to use standard relaxation to find its minimum. We thus first minimize the Rosetta energy function in the base space of the flow which is then followed by standard Rosetta Relax.

Figure 6 shows that diffeomorphic optimization leads to substantially lower energy values throughout the pdb test set. Furthermore, the energy values of the baseline Rosetta Relax do not improve significantly by running a longer optimization or using more seeds. Thus, the relative advantage is not a function of the higher cost of diffeomorphic optimization. Further analysis is provided in Appendix H.4.

Lattice Gauge Theory: lattice gauge theory offers what the protein experiments cannot, namely exact criteria that decide whether an optimized sample still shares the structure of the data. We first consider a two-dimensional $SU(3)$ theory on an 8×8 lattice for which we train a diffusion model on Boltzmann samples at $\beta = 8$ and minimize the Wilson action itself. In two dimensions the plaquettes are statistically independent, so the 2×2 Wilson loop factorizes and the ratio $R_{2\times2}$ equals one at every β without any free parameter. Figure 8a shows that diffeomorphic optimization stays within 11% of this value along the entire descent and ends at 0.95, which is the resolution of the generative model itself whose unoptimized samples give 0.967 ± 0.015. Direct gradient descent on the links, a discretization of the Wilson flow, instead breaks the identity down to 0.59. We then consider a two-dimensional $U(1)$ theory on a 32×32 lattice, where the topological charge Q labels sectors that a continuous path can only leave by creating a vortex. Asked to move one sector up, diffeomorphic optimization places 98% of the configurations in the target sector while its vortex count only decreases, from 7.3 to below 0.02. Direct gradient descent reaches 15% at the same loss level and does so by building vortices, increasing their number from 7.8 to 8.5, see Figure 8b. Further analysis is provided in Appendix H.5.

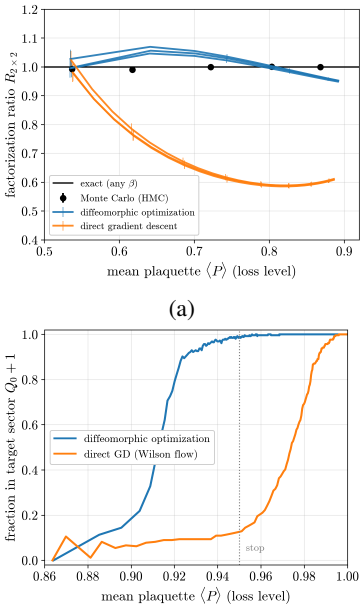


Figure 8: Diffeomorphic optimization on two-dimensional lattice gauge theories. **(a)** Factorization ratio $R_{2\times2}$ for $SU(3)$ against the loss level $\langle P \rangle$; the black line is the exact value at every β, the dots are Monte Carlo samples. **(b)** Fraction of $U(1)$ configurations in the target sector $Q_0 + 1$; the dotted line marks the stopping level.

6 Conclusion and Limitations

We have proposed diffeomorphic optimization which allows us to minimize arbitrary differentiable cost functions on the data manifold. This is achieved with off-the-shelf autograd engines with simple gradient wrappers making it a readily available tool for practitioners. A downside of this approach are the considerable numerical costs due to the backpropagation through the generation. As argued in the introduction, these costs are of lower concern in protein design, where experimental wet-lab verification rather than sampling is the bottleneck. The method also inherits the support of the generative model and cannot reach designs to which the model assigns no mass, the flip side of remaining on the learned manifold. Nothing constrains the base point to regions of high base-space density either, so large learning rates can drift towards implausible samples, which is why we recommend monitoring an independent criterion alongside the optimized objective.

REFERENCES

Josh Abramson, Jonas Adler, Jack Dunger, Richard Evans, Tim Green, Alexander Pritzel, Olaf Ronneberger, Lindsay Willmore, Andrew J Ballard, Joshua Bambrick, et al. Accurate structure prediction of biomolecular interactions with alphafold 3. *Nature*, 630(8016):493–500, 2024.

Michael S. Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. *arXiv preprint arXiv:2209.15571*, 2022. URL https://arxiv.org/abs/2209.15571.

Rebecca F Alford, Andrew Leaver-Fay, Jeliazko R Jeliazkov, Matthew J O'Meara, Frank P DiMaio, Hahnbeom Park, Maxim V Shapovalov, P Douglas Renfrew, Vikram K Mulligan, Kalli Kappel, et al. The rosetta all-atom energy function for macromolecular modeling and design. *Journal of chemical theory and computation*, 13(6):3031–3048, 2017.

Ivan Anishchenko, Samuel J Pellock, Tamuka M Chidyausiku, Theresa A Ramelot, Sergey Ovchinnikov, Jingzhou Hao, Khushboo Bafna, Christoffer Norn, Alex Kang, Asim K Bera, et al. De novo protein design by deep network hallucination. *Nature*, 600(7889):547–552, 2021.

Simone Bacchio, Pan Kessel, Stefan Schaefer, and Lorenz Vaitl. Learning trivializing gradient flows for lattice gauge theories. *Physical Review D*, 107(5):L051504, 2023.

Arpit Bansal, Hong-Min Chu, Avi Schwarzschild, Soumyadip Sengupta, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Universal guidance for diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 843–852, 2023.

Heli Ben-Hamu, Omri Puny, Itai Gat, Brian Karrer, Uriel Singer, and Yaron Lipman. D-flow: Differentiating through flows for controlled generation. *arXiv preprint arXiv:2402.14017*, 2024.

Nathaniel R Bennett, Joseph L Watson, Robert J Ragotte, Andrew J Borst, Déjenaé L See, Connor Weidle, Riti Biswas, Ellen L Shrock, Philip JY Leung, Buwei Huang, et al. Atomically accurate de novo design of single-domain antibodies. *biorxiv*, 2024.

Bradley CA Brown, Anthony L Caterini, Brendan Leigh Ross, Jesse C Cresswell, and Gabriel Loaiza-Ganem. Verifying the union of manifolds hypothesis for image data. *arXiv preprint arXiv:2207.02862*, 2022.

Gabriel Cardoso, Yazid Janati El Idrissi, Sylvain Le Corff, and Eric Moulines. Monte carlo guided diffusion for bayesian linear inverse problems. *arXiv preprint arXiv:2308.07983*, 2023.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Yehlin Cho, Martin Pacesa, Zhidian Zhang, Bruno Correia, and Sergey Ovchinnikov. Boltzdesign1: Inverting all-atom structure prediction model for generalized biomolecular binder design. *bioRxiv*, pp. 2025–04, 2025.

Kevin Clark, Paul Vicol, Kevin Swersky, and David J Fleet. Directly fine-tuning diffusion models on differentiable rewards. *arXiv preprint arXiv:2309.17400*, 2023.

Gabriele Corso, Hannes Stärk, Bowen Jing, Regina Barzilay, and Tommi Jaakkola. Diffdock: Diffusion steps, twists, and turns for molecular docking. In *International Conference on Learning Representations (ICLR)*, 2023.

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

Amit Dhurandhar, Pin-Yu Chen, Ronny Luss, Chun-Chen Tu, Paishun Ting, Karthikeyan Shanmugam, and Payel Das. Explanations based on the missing: Towards contrastive explanations with pertinent negatives. *Advances in neural information processing systems*, 31, 2018.

Ann-Kathrin Dombrowski, Jan E Gerken, and Pan Kessel. Diffeomorphic explanations with normalizing flows. In *ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models*, 2021.

Ann-Kathrin Dombrowski, Jan E Gerken, Klaus-Robert Müller, and Pan Kessel. Diffeomorphic counterfactuals with generative models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(5):3257–3274, 2023.

Zehao Dou and Yang Song. Diffusion posterior sampling for linear inverse problem solving: A filtering perspective. In *The Twelfth International Conference on Learning Representations*, 2024.

Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. Testing the manifold hypothesis. *Journal of the American Mathematical Society*, 29(4):983–1049, 2016.

Nathan C Frey, Isidro Hötzel, Samuel D Stanton, Ryan Kelly, Robert G Alberstein, Emily Makowski, Karolis Martinkus, Daniel Berenberg, Jack Bevers III, Tyler Bryson, et al. Lab-in-the-loop therapeutic antibody design with deep learning. *bioRxiv*, pp. 2025–02, 2025.

Casper A Goverde, Benedict Wolf, Hamed Khakzad, Stéphane Rosset, and Bruno E Correia. De novo protein design by inversion of the alphafold structure prediction network. *Protein Science*, 32 (6):e4653, 2023.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

Qiuyue Hu, Zechen Wang, Jintao Meng, Weifeng Li, Jingjing Guo, Yuguang Mu, Sheng Wang, Zheng Liangzhen, and Yanjie Wei. OpenDock: A pytorch-based open-source framework for protein-ligand docking and modelling. *Bioinformatics*, pp. btae628, 10 2024. ISSN 1367-4811. doi: 10.1093/bioinformatics/btae628. URL https://doi.org/10.1093/bioinformatics/btae628.

Bowen Jing, Bonnie Berger, and Tommi Jaakkola. Alphafold meets flow matching for generating protein ensembles. In *Forty-first International Conference on Machine Learning*, 2024.

Shalmali Joshi, Oluwasanmi Koyejo, Warut Vijitbenjaronk, Been Kim, and Joydeep Ghosh. Towards realistic individual recourse and actionable explanations in black-box decision making systems. *arXiv preprint arXiv:1907.09615*, 2019.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589, 2021.

Bobak Kiani, Jason Wang, and Melanie Weber. Hardness of learning neural networks under the manifold hypothesis. *Advances in Neural Information Processing Systems*, 37:5661–5696, 2024.

Patrick Kidger, James Foster, Xuechen Chen Li, and Terry Lyons. Efficient and accurate gradients for neural sdes. *Advances in Neural Information Processing Systems*, 34:18747–18761, 2021.

David E Kim, Ben Blum, Philip Bradley, and David Baker. Sampling bottlenecks in de novo protein structure prediction. *Journal of molecular biology*, 393(1):249–260, 2009.

Takatsugu Kosugi and Masahito Ohue. Solubility-aware protein binding peptide design using alphafold. *Biomedicines*, 10(7):1626, 2022.

Andrew Leaver-Fay, Jeff Flatten, Alex Ford, Joseph Kleinhenz, David Solberg, Henry amd Baker, Andrew M Watkins, Brian Kuhlman, and Frank DiMaio. tmol: a gpu-accelarated, pytorch implementation of rosetta's relax protocol (manuscript in preparation), 2025. URL https://github.com/uw-ipd/tmol.

John M Lee. *Introduction to Riemannian manifolds*, volume 2. Springer, 2018.

Xuechen Li, Ting-Kam Leonard Wong, Ricky TQ Chen, and David K Duvenaud. Scalable gradients and variational inference for stochastic differential equations. In *Symposium on Advances in Approximate Bayesian Inference*, pp. 1–28. PMLR, 2020.

Yaron Lipman, Marton Havasi, Peter Holderrieth, Neta Shaul, Matt Le, Brian Karrer, Ricky TQ Chen, David Lopez-Paz, Heli Ben-Hamu, and Itai Gat. Flow matching guide and code. *arXiv preprint arXiv:2412.06264*, 2024.

Xingchao Liu, Lemeng Wu, Shujian Zhang, Chengyue Gong, Wei Ping, and Qiang Liu. Flowgrad: Controlling the output of generative odes with gradients. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 24335–24344, 2023.

Aaron Lou, Derek Lim, Isay Katsman, Leo Huang, Qingxuan Jiang, Ser Nam Lim, and Christopher M De Sa. Neural manifold ordinary differential equations. *Advances in Neural Information Processing Systems*, 33:17548–17558, 2020.

Emile Mathieu and Maximilian Nickel. Riemannian continuous normalizing flows. *Advances in Neural Information Processing Systems*, 33:2503–2515, 2020.

Martin Pacesa, Lennart Nickel, Christian Schellhaas, Joseph Schmidt, Ekaterina Pyatova, Lucas Kissling, Patrick Barendse, Jagrity Choudhury, Srajan Kapoor, Ana Alcaraz-Serna, et al. Bindcraft: one-shot design of functional protein binders. *bioRxiv*, pp. 2024–09, 2024.

Danilo Jimenez Rezende and Shakir Mohamed. Normalizing flows on tori and spheres. In *International Conference on Machine Learning*, pp. 8083–8092, 2020.

Raghav Singhal, Zachary Horvitz, Ryan Teehan, Mengye Ren, Zhou Yu, Kathleen McKeown, and Rajesh Ranganath. A general framework for inference-time scaling and steering of diffusion models. *arXiv preprint arXiv:2501.06848*, 2025.

Brian L Trippe, Jason Yim, Doug Tischer, David Baker, Tamara Broderick, Regina Barzilay, and Tommi Jaakkola. Diffusion probabilistic modeling of protein backbones in 3d for the motif-scaffolding problem. *arXiv preprint arXiv:2206.04119*, 2022.

Oleg Trott and Arthur J Olson. Autodock vina: Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *Journal of Computational Chemistry*, 31(2):455–461, 2010. doi: 10.1002/jcc.21334.

Luran Wang, Chaoran Cheng, Yizhen Liao, Yanru Qu, and Ge Liu. Training free guided flow matching with optimal control. *arXiv preprint arXiv:2410.18070*, 2024.

Renxiao Wang, Xueliang Fang, Yipin Lu, Chao-Yie Yang, and Shaomeng Wang. The pdbbind database: methodologies and updates. *Journal of medicinal chemistry*, 48(12):4111–4119, 2005.

Joseph L Watson, David Juergens, Nathaniel R Bennett, Brian L Trippe, Jason Yim, Helen E Eisenach, Woody Ahern, Andrew J Borst, Robert J Ragotte, Lukas F Milles, et al. De novo design of protein structure and function with rfdiffusion. *Nature*, 620(7976):1089–1100, 2023.

René T Wintjens, Marianne J Rooman, and Shoshana J Wodak. Automatic classification and analysis of $\alpha\alpha$-turn motifs in proteins. *Journal of molecular biology*, 255(1):235–253, 1996.

Luhuan Wu, Brian Trippe, Christian Naesseth, David Blei, and John P Cunningham. Practical and asymptotically exact conditional sampling in diffusion models. *Advances in Neural Information Processing Systems*, 36:31372–31403, 2023.

Jason Yim, Andrew Campbell, Andrew YK Foong, Michael Gastegger, José Jiménez-Luna, Sarah Lewis, Victor Garcia Satorras, Bastiaan S Veeling, Regina Barzilay, Tommi Jaakkola, et al. Fast protein backbone generation with se (3) flow matching. *arXiv preprint arXiv:2310.05297*, 2023a.

Jason Yim, Brian L Trippe, Valentin De Bortoli, Emile Mathieu, Arnaud Doucet, Regina Barzilay, and Tommi Jaakkola. Se (3) diffusion model with application to protein backbone generation. *arXiv preprint arXiv:2302.02277*, 2023b.

A Riemannian Gradient Implementation for $\mathbf{SO(3)}$

As derived in Theorem 2, we can efficiently modify existing autograd engines to calculate the $SO(3)$ Riemannian gradient. Specifically, we can wrap the tensor by an operation that acts like the identity in the forward pass and adjusts the gradient to coincide with the Riemannian gradient in the backward pass using Theorem 2. In PyTorch, this can most conveniently be implemented with a backward hook as shown below. We emphasize that this implementation is completely general and ensures that the Riemannian gradient is seamlessly integrated in existing autograd functionality.

```
import torch
from scipy.spatial.transform import Rotation

def so3tensor(*args, **kwargs):
    x = torch.tensor(*args, **kwargs)

    def so3_backward_hook(grad):
        so3_grad = grad @ x.transpose(-1, -2)
        return so3_grad - so3_grad.transpose(-1, -2)

    if x.requires_grad:
        x.register_hook(so3_backward_hook)
    return x

x = so3tensor(Rotation.random(1).as_matrix(), requires_grad=True)
y = so3tensor(Rotation.random(1).as_matrix())

loss = (x - y).pow(2).sum()
so3_grad = torch.autograd.grad(loss, x)[0]
print(so3_grad)

>>tensor([[[ 0.0000,  0.8490,  0.1187],
           [-0.8490,  0.0000, -2.2494],
           [-0.1187,  2.2494,  0.0000]]], dtype=torch.float64)
```

B Expanded Related Work

Backpropagation through folding models: protein hallucination is a version of computational protein design which uses backpropagation through a folding model to its input sequence Anishchenko et al. (2021); Kosugi & Ohue (2022); Goverde et al. (2023); Pacesa et al. (2024). Interestingly, even the most recent work Cho et al. (2025) based on AlphaFold3 avoids explicit backpropagation through its diffusion structure module and instead relies on the pair representation - a limitation that could potentially be overcome by our proposed methods.

Gradient descent and normalizing flows has been explored in the explainability literature to generate counterfactual explanations Joshi et al. (2019); Dombrowski et al. (2021; 2023); Dhurandhar et al. (2018) although not for flow matching and diffusion models. Ben-Hamu et al. (2024) proposes to differentiate through flows for controlled generation. Our work builds on this reference by generalizing it to matrix Lie groups, which is of high relevance for proteins. We also provide a detailed theoretical analysis of the method and derive an efficient adjoint state method. Wang et al. (2024) explores related ideas for matrix groups in the framework of optimal control. Specifically, the authors propose to add an additive control to the vector field of the flow. Our approach does not use a control but rather optimizes the initial condition following Ben-Hamu et al. (2024). We discuss the relationship to this reference in more detail in the appendix and compare in detailed numerical experiments to their approach. Liu et al. (2023) similarly relies on control variables but directly applies gradient descent to them. This reference does however not consider matrix groups.

Adjoint state method on manifolds: the adjoint state method on manifolds has been discussed in other works - however only in terms of charts Lou et al. (2020); Mathieu & Nickel (2020) or particular manifolds Rezende & Mohamed (2020); Bacchio et al. (2023); Albergo & Vanden-Eijnden (2022). The adjoint state method derived in our work is applicable to any matrix Lie group and is

particularly efficient for the group $SE(3)$ which plays a crucial role the frame-based protein backbone representation.

Guidance: a widely used method to bias diffusion and flow-matching models towards certain desiderata is guidance. There exist various flavors of it, such as classifier-based guidance Dhariwal & Nichol (2021), classifier-free guidance Ho & Salimans (2022), and universal guidance Bansal et al. (2023). There has also been substantial recent interest in biasing the generation with additional reward functions Wu et al. (2023); Trippe et al. (2022); Cardoso et al. (2023); Dou & Song (2024); Singhal et al. (2025) often relying on sequential Monte Carlo. However, these methods struggle for guidance potentials that are sensitive to fine-grained details of the final sample, such as force fields or certain biochemical properties, as they tend to rely on few-shot denoising and require carefully hyperparameter finetuing of their starting point and time-dependent weighting factor. We discuss the limitations of guidance in more detail in Appendix F.

Relation to Wang et al. (2024): a closely related reference is Wang et al. (2024) which outlines a very nice approach to maximize a terminal reward using deterministic optimal control. Specifically, the authors propose to add an additive control θ_τ to the vector field of the flow

$$\frac{dx_\tau}{d\tau} = v_\theta(x_\tau, \tau) \qquad \rightarrow \qquad \frac{dx_\tau}{d\tau} = v_\theta(x_\tau, \tau) + \theta_\tau \tag{20}$$

The authors then propose to optimize the control with respect to a (regularized) loss

$$\mathcal{L}(x_1^\theta) + d(x_1^\theta, x_1^0) + \frac{1}{2}\int_0^1 d\tau \, ||\theta_\tau||^2 \tag{21}$$

where x_1^θ corresponds to the terminal value under the control θ_τ and x_1^0 is the terminal value for no control. Furthermore, $d(\cdot, \cdot)$ denotes a distance.

The optimal control approach defines control terms θ_τ at intermediate time steps τ during the generative process. By construction, the control thus acts upon the generation not only at the initial condition $\tau = 0$ but also for $\tau > 0$. A change in initial condition $x_0 \to x_0 + \delta x_0$ can then only be interpreted as a control if the latter is not a function $\forall\tau$ but rather concentrates all modelling capacity at only $\tau = 0$. In the continuous time setting this would imply a control proportional to a Dirac impulse $\delta(\tau)$ in time, seen as follows

$$x_1 = x_0 + \delta x_0 + \int_0^1 v(x_\tau) d\tau = x_0 + \int_0^1 \left(v_\theta(x_\tau) + \delta(\tau)\delta x_0\right) d\tau \tag{22}$$

and thus implying a distributional control $\theta_\tau = \delta(\tau)\delta x_0$. Note that this distributional control has highly undesirable properties from a numerics standpoint, i.e., it is necessarily divergent for $\tau = 0$ and vanishing for all other values of flow time τ. Intuitively, it would require the control to voluntarily constrain all its modelling capacity to the time of the initial condition $\tau = 0$, something that is hard to achieve in practice with unconstrained optimization.

Diffusion and flow models are trained to map a simple base space variable to a target distribution. Changing the base space variable ensures that the samples remain on the learned manifold in the target space. This can not be guaranteed if the trajectory is changed at intermediate points. The control parameters can exploit the loss function $\mathcal{L}(x_1^\theta)$ while moving the sample off manifold. To counteract such movements, a distance loss term $d(x_1^\theta, x_1^0)$ in the target space is applied but this but simultaneously constraints how much the original loss function can be optimized. Diffeomorphic optimization solves both problems conveniently at the same time by allowing free, unconstrained movement over the manifold while also remaining on the learned manifold at all times.

C Proofs

C.1 Proof of Theorem 1

Lemma 1. *For $g : M \to N$, it holds that*

$$g(\exp_p(\lambda v)) = \exp_{g(p)}(\lambda \, dg_p(v) + \mathcal{O}(\lambda^2)) \tag{23}$$

for any $p \in M$, $v \in T_pM$, and small $\lambda \in \mathbb{R}$.

Proof. For proving this statement, it is useful to consider normal coordinates. For $p \in M$ and a neighborhood U of $0 \in T_pM$, these coordinates are given by the chart $\psi : U \subset T_pM \to M$ with $\psi(v) = \exp_p(v)$. The coordinates y^α of a point $q \in M$ in this chart are obtained by

$$q = \exp_p(y^\alpha e_\alpha) \tag{24}$$

where e_α is an orthonormal basis of T_pM and we have used the Einstein summation convention. This implies that

$$y^\alpha(\exp_q(v)) = v^\alpha \tag{25}$$

Using normal coordinates for both M and N, we can expand the function g using Taylor's theorem

$$g(\widehat{\exp_p(\lambda v)}) = g^\alpha(y^\beta) = g^\alpha(\lambda v^\beta) = g^\alpha(0) + \lambda \frac{\partial g^\alpha}{\partial y^\beta} v^\beta + \mathcal{O}(\lambda^2) = \widehat{\exp_{g(p)}}(\lambda \widehat{dg_p}(v) + \mathcal{O}(\lambda^2)) \tag{26}$$

where we have used that the connection Γ vanishes at the origin of normal coordinates, which suffices at first order and the hat symbol denotes the coordinate representation. Since the left and the right hand side are written in terms of covariant objects, the result stated in the theorem follows. □

Lemma 2. *Let $g : Z \to X$ be a diffeomorphism with Z being a Riemannian manifold with metric G. Then, it holds that*

$$dg_z(\mathit{grad}_z^G \mathcal{L} \circ g) = \mathit{grad}_{g(z)}^{\tilde{G}} \mathcal{L} \tag{27}$$

where $\tilde{G} = g_* G$ is the pushforward metric of G.

Proof. This statement is easily shown in coordinates. Let z^μ denote coordinates on Z and $x^\alpha = g^\alpha(z)$. Then, the coordinate representation of the pushforward metric $\tilde{G}_{\alpha\beta}$ follows by

$$G_{\mu\nu} dz^\mu dz^\nu = G_{\mu\nu} \frac{\partial z^\mu}{\partial x^\alpha} \frac{\partial z^\nu}{\partial x^\beta} dx^\alpha dx^\beta = \tilde{G}_{\alpha\beta} dx^\alpha dx^\beta \tag{28}$$

In these coordinates, $dg_z(\mathrm{grad}_z^G \mathcal{L} \circ g)$ is given by

$$\frac{\partial x^\alpha}{\partial z^\mu} G^{\mu\nu} \frac{\partial (\mathcal{L} \circ g)}{\partial z^\nu} = \frac{\partial x^\alpha}{\partial z^\mu} G^{\mu\nu} \frac{\partial x^\sigma}{\partial z^\nu} \frac{\partial \mathcal{L}}{\partial x^\sigma} = \tilde{G}^{\alpha\sigma} \frac{\partial \mathcal{L}}{\partial x^\sigma} \tag{29}$$

which corresponds to the coordinate representation of the right hand side. □

Theorem 1 follows almost immediately using the the two lemmas:

Proof. By Lemma 1, it follows that

$$g(\exp_z(-\lambda \,\mathrm{grad}_z^G \mathcal{L} \circ g)) = \exp_{g(z)}(-\lambda \, dg_z(\mathrm{grad}_z^G \mathcal{L}) + \mathcal{O}(\lambda^2)) \,. \tag{30}$$

The right hand side can then be rewritten by Lemma 2 as

$$\exp_{g(z)}(-\lambda \, dg_z(\mathrm{grad}_z^G \mathcal{L}) + \mathcal{O}(\lambda^2)) = \exp_{g(z)}(-\lambda \,\mathrm{grad}_{g(z)}^{\tilde{G}} \mathcal{L} + \mathcal{O}(\lambda^2)) \tag{31}$$

which shows the stated result. □

C.2 Proof of Theorem 2

We consider the definition of the Riemannian gradient on the Lie algebra

$$\partial^a f(R) = \frac{d}{d\tau} f(\exp(\tau T^a)R)|_{\tau=0} \tag{32}$$

$$= \frac{df}{dR_{\alpha\beta}} \frac{d(\exp(\tau T^a)R)_{\alpha\beta}}{d\tau}|_{\tau=0} \tag{33}$$

$$= \frac{df}{dR_{\alpha\beta}} (T^a R)_{\alpha\beta} \tag{34}$$

$$= \frac{df}{dR_{\alpha\beta}} T^a_{\alpha\sigma} R_{\sigma\beta} \tag{35}$$

$$= \frac{df}{dR_{\alpha\beta}} T^a_{\alpha\sigma} (R^T)_{\beta\sigma} \qquad | \quad T^a = -(T^a)^T \tag{36}$$

$$= -\frac{df}{dR_{\alpha\beta}} T^a_{\sigma\alpha} (R^T)_{\beta\sigma} \tag{37}$$

$$= -\text{tr}\left(T^a \frac{df}{dR} R^T \right) \tag{38}$$

$$= -\frac{1}{2}\text{tr}\left(T^a \, 2\frac{df}{dR} R^T \right) \tag{39}$$

$$= \langle T^a, 2\frac{df}{dR} R^T \rangle \tag{40}$$

where we have used the Einstein summation convention and the definition of inner product $\langle A, B\rangle = \frac{1}{2}\text{tr}A^T B$ for Lie algebra elements $A, B \in \mathfrak{so}(3)$. Recall that the Riemannian gradient is given by

$$\partial f \equiv \sum_a T^a \partial_a f(R) \tag{41}$$

Due to the orthonormality of the generators, $\langle T^a, T^b\rangle = \delta^{ab}$ and denoting the antisymmetric part $[M]_A = \frac{1}{2}(M - M^T)$ of a matrix M, it thus holds that

$$\partial f = 2\left[\frac{df}{dR} R^T \right]_A , \tag{42}$$

The antisymmetric component arises in this step because for an arbitrary matrix M, we have $\langle T^a, M\rangle = \langle T^a, [M]_A\rangle$.

C.3 Proof of Theorem 3

The adjoint state is given by

$$A_\tau = \nabla_{Z_\tau} \mathcal{L} \tag{43}$$

for which we want to derive a differential equation for its time evolution.

For this, we define the time evolution operator

$$T_\epsilon : SO(3) \to SO(3)\,, \qquad Z_\tau \mapsto Z_{\tau+\epsilon} = \exp(\epsilon V_\theta(Z_\tau))\, Z_\tau\,. \tag{44}$$

We use the chain rule equation 99 to derive that

$$A^a_\tau = \nabla^a_{Z_\tau} \mathcal{L} = \sum_b \nabla^b_{Z_{\tau+\epsilon}} \mathcal{L}\, \mathfrak{D}^{ba} T_\epsilon \tag{45}$$

$$= \sum_b A^b_{\tau+\epsilon}\, \mathfrak{D}^{ba} T_\epsilon\,. \tag{46}$$

The differential equation 98 of the time evolution operator is given by:

$$\mathfrak{D}^{ba} T_\epsilon = \langle T^b, \nabla^a_{Z_\tau} T_\epsilon\, T_\epsilon^\top \rangle \tag{47}$$

$$= \langle T^b, \nabla^a_{Z_\tau} (e^{\epsilon V_\theta(Z_\tau)} Z_\tau)\; Z_\tau^\top e^{-\epsilon V_\theta(Z_\tau)} \rangle\,, \tag{48}$$

where we have used that $V_\theta \in \mathfrak{so}(3)$ is antisymmetric. We now expand this expression up to quadratic order in the step size ϵ to obtain

$$\mathfrak{D}^{ba} T_\epsilon = \langle T^b, \nabla^a_{Z_\tau}(\ (\mathbb{I} + \epsilon V_\theta(Z_\tau))\, Z_\tau)\ Z_\tau^\top (\mathbb{I} - \epsilon V_\theta(Z_\tau))\,\rangle + \mathcal{O}(\epsilon^2) \tag{49}$$

$$= \delta^{ba} + \epsilon \langle T^b, [V_\theta, T^a]\rangle + \epsilon \langle T^b, \nabla^a_{Z_\tau} V_\theta(Z_\tau)\rangle + \mathcal{O}(\epsilon^2) \tag{50}$$

We now use the fact that

$$\langle T^b, [V_\theta, T^a]\rangle = -\frac{1}{2}\mathrm{tr}\, T^b\,(V_\theta T^a - T^a V_\theta) = -\frac{1}{2}\mathrm{tr}\, T^a\left(T^b V_\theta - V_\theta T^b\right) = \langle T^a, [T^b, V_\theta]\rangle\,. \tag{51}$$

Expanding $V_\theta = \sum_c V_\theta^c T^c$ and similarly for $[T^b, V_\theta] \in \mathfrak{so}(3)$, we then obtain that the differential is given by

$$\mathfrak{D}^{ba} T_\epsilon = \delta^{ba} - \epsilon [V_\theta, T^b]^a + \epsilon \nabla^a_{Z_\tau} V_\theta(Z_\tau)^b + \mathcal{O}(\epsilon^2) \tag{52}$$

Plugging this result into the chain rule for the adjoint state equation 46, we obtain

$$A^a_\tau = A^a_{\tau+\epsilon} - \epsilon [V_\theta, A_{\tau+\epsilon}]^a + \epsilon \sum_b A^b_{\tau+\epsilon}\, \nabla^a_{Z_\tau} V_\theta(Z_\tau)^b + \mathcal{O}(\epsilon^2)\,. \tag{53}$$

We now rearrange this to isolate the finite time difference on the right-hand-side

$$\tfrac{1}{\epsilon}\left(A^a_{\tau+\epsilon} - A^a_\tau\right) = [V_\theta, A_{\tau+\epsilon}]^a - \sum_b A^b_{\tau+\epsilon}\, \nabla^a_{Z_\tau} V_\theta(Z_\tau)^b + \mathcal{O}(\epsilon)\,. \tag{54}$$

Taking the limit $\epsilon \to 0$ gives

$$\frac{d}{d\tau} A^a_\tau = [V_\theta, A_\tau]^a - \sum_b A^b_\tau\, \nabla^a_{Z_\tau} V_\theta(Z_\tau)^b\,. \tag{55}$$

Multiplying by the generator T^a and summing over the index a, we then obtain:

$$\frac{d}{d\tau} A_\tau = [V_\theta, A_\tau] - \sum_b A^b_\tau\, \nabla_{Z_\tau} V_\theta(Z_\tau)^b\,, \tag{56}$$

which is the claimed time-evolution equation of the theorem.

C.4 Coupled Adjoint System on $SE(3)$

Theorem 3 assumes a rotational velocity field $V_\theta(Z_\tau)$ that does not depend on the translation. For the $SE(3)$ flow equation 14, both velocity fields depend on Z_τ and z_τ, so that the adjoint states of rotation and translation are coupled.

Theorem 4. *The gradients of the reparameterized loss function*

$$\nabla_Z \mathcal{L}(g(Z,z)) = A_0\,, \qquad \frac{d\mathcal{L}(g(Z,z))}{dz} = a_0 \tag{57}$$

can be obtained by the following coupled terminal value problem for the Lie-algebra-valued adjoint state $A_\tau = \sum_{a=1}^3 A^a_\tau T^a \in \mathfrak{so}(3)$ and the adjoint state $a_\tau \in \mathbb{R}^3$:

$$\frac{dA_\tau}{d\tau} = [V_\theta, A_\tau] - \sum_{b=1}^3 A^b_\tau \nabla_{Z_\tau} V^b_\theta - \sum_{i=1}^3 a^i_\tau \nabla_{Z_\tau} v^i_\theta\,, \qquad A_1 = \nabla_R \mathcal{L}(R,t)\,, \tag{58}$$

$$\frac{da_\tau}{d\tau} = -\sum_{i=1}^3 a^i_\tau \frac{dv^i_\theta}{dz_\tau} - \sum_{b=1}^3 A^b_\tau \frac{dV^b_\theta}{dz_\tau}\,, \qquad a_1 = \frac{d\mathcal{L}(R,t)}{dt}\,, \tag{59}$$

where $V_\theta = V_\theta(Z_\tau, z_\tau)$, $v_\theta = v_\theta(Z_\tau, z_\tau)$ and $\tau \in [0,1]$.

Proof: we extend the time evolution operator of Appendix C.3 to both components,

$$T_\epsilon : SE(3) \to SE(3)\,, \quad (Z_\tau, z_\tau) \mapsto (Z_{\tau+\epsilon}, z_{\tau+\epsilon}) = \left(e^{\epsilon V_\theta(Z_\tau, z_\tau)} Z_\tau,\ z_\tau + \epsilon\, v_\theta(Z_\tau, z_\tau)\right)\,. \tag{60}$$

Since $Z_{\tau+\epsilon}$ and $z_{\tau+\epsilon}$ both depend on Z_τ and z_τ, the chain rule equation 99 contains two contributions for each adjoint state:

$$A^a_\tau = \sum_b A^b_{\tau+\epsilon}\, \mathfrak{D}^{ba} T_\epsilon + \sum_i a^i_{\tau+\epsilon}\, \nabla^a_{Z_\tau} z^i_{\tau+\epsilon}\,, \tag{61}$$

$$a^j_\tau = \sum_i a^i_{\tau+\epsilon}\, \frac{dz^i_{\tau+\epsilon}}{dz^j_\tau} + \sum_b A^b_{\tau+\epsilon}\, \langle T^b, \tfrac{d}{dz^j_\tau} Z_{\tau+\epsilon}\; Z^\top_{\tau+\epsilon}\rangle\,. \tag{62}$$

The first term in the first line is the one evaluated in Appendix C.3, now with V_θ evaluated at (Z_τ, z_τ). The remaining derivatives are, to first order in ϵ,

$$\nabla^a_{Z_\tau} z^i_{\tau+\epsilon} = \epsilon\, \nabla^a_{Z_\tau} v^i_\theta + \mathcal{O}(\epsilon^2)\,, \tag{63}$$

$$\frac{dz^i_{\tau+\epsilon}}{dz^j_\tau} = \delta^{ij} + \epsilon\, \frac{dv^i_\theta}{dz^j_\tau} + \mathcal{O}(\epsilon^2)\,, \tag{64}$$

$$\langle T^b, \tfrac{d}{dz^j_\tau}(e^{\epsilon V_\theta} Z_\tau)\; Z^\top_\tau e^{-\epsilon V_\theta}\rangle = \epsilon\, \frac{dV^b_\theta}{dz^j_\tau} + \mathcal{O}(\epsilon^2)\,. \tag{65}$$

Substituting these expressions, isolating the finite time differences and taking the limit $\epsilon \to 0$ as in Appendix C.3 yields the claimed equations; the terminal conditions follow from $(Z_1, z_1) = (R, t)$. Theorem 3 and the translational adjoint of Section 4.1 are recovered when V_θ does not depend on z_τ and v_θ does not depend on Z_τ.

D Lightning Review of Differential Geometry and Lie Groups

We first review the concepts of Riemannian geometry that are used in Section 3 and then specialize them to Lie groups.

Manifolds and coordinates: a manifold $\mathcal{M}$ is a space that locally takes the form of $\mathbb{R}^D$, similar to the earth which can locally approximated by flat three-dimensional space. More formally, for each point $p \in \mathcal{M}$, there exists a chart $\varphi : U \to \mathbb{R}^D$ where U is an open subset of $\mathcal{M}$ containing the point p. In this sense, the manifold is locally described by a Euclidian D space. The pair (U, φ) is called coordinate chart and the component functions x^i of $\varphi(p) = (x^1(p), \cdots, x^D(p))$ are called coordinates.

Tangent space: the tangent space $T_p\mathcal{M}$ contains the velocity vectors $\frac{d}{dt}\gamma(t)|_{t=0}$ of curves $\gamma : \mathbb{R} \to \mathcal{M}$ with $\gamma(0) = p$. It can be shown that the tangent space $T_p\mathcal{M}$ is a D-dimensional vector space. Let (U, φ) be a coordinate chart on $\mathcal{M}$ with coordinates x. We can then define $\varphi \circ \lambda_k(t) = (x^1(p), \ldots, x^k(p)+t, \ldots, x^D(p))$ with $k \in \{1, \ldots, D\}$. This implicitly defines curves $\lambda_k : \mathbb{R} \to \mathcal{M}$ through p. We denote the corresponding tangent vectors as $\frac{\partial}{\partial x^k} = \frac{d}{dt}\lambda_k(t)|_{t=0}$ and it can be shown that they form a basis of the tangent space $T_p\mathcal{M}$. The differential $dg_p : T_p\mathcal{M} \to T_{f(p)}\mathcal{N}$ for a map $g : \mathcal{M} \to \mathcal{V}$ between two manifolds $\mathcal{M}$ and $\mathcal{V}$ is a linear map between the respective tangent spaces. The curve $\gamma_v : \mathbb{R} \to \mathcal{M}$, corresponding to the vector $v \in T_p\mathcal{M}$, is mapped by the differential dg_p to the curve $g \circ \gamma_v : \mathbb{R} \to \mathcal{N}$. Thus, the differential acts as $dg_p[v] = \frac{d}{dt} g \circ \gamma_v(t)|_{t=0}$.

Riemannian metric: A Riemannian manifold is endowed with an inner product $\langle \cdot, \cdot \rangle_{G_p} : T_p\mathcal{M} \times T_p\mathcal{M} \to \mathbb{R}$ for each tangent space, which allows us to define a notion of the length of tangent vectors. G is also known as the Riemannian metric. We refer to the excellent textbook Lee (2018).

Riemannian gradient: The Riemannian gradient $\operatorname{grad}^G_p f \in T_p\mathcal{M}$ at point $p \in \mathcal{M}$ of a function $f : \mathcal{M} \to \mathbb{R}$ is the vector uniquely defined by the relation

$$\forall v \in T_p\mathcal{M} : \qquad \langle v, \operatorname{grad}^G_p f\rangle_{G_p} = df_p[v] \tag{66}$$

and crucially depends on Riemannian metric G of the manifold. In coordinates, the Riemannian gradient is $\widehat{\operatorname{grad}^G_p f} = \sum_j G^{ij}\partial_j f$ where G^{ij} is the inverse of the metric tensor $G_{ij} = G(\frac{\partial}{\partial x^i}, \frac{\partial}{\partial x^j})$.

Exponential map: On a Riemannian manifold, we can further define geodesics, i.e. curves $\gamma : I \subset \mathbb{R} \to \mathcal{M}$ that have vanishing acceleration with respect to the covariant derivative induced by the Riemannian connection. For each $p \in \mathcal{M}$ and $v \in T_p\mathcal{M}$, there exists a unique geodesic γ_v that satisfies $\gamma_v(0) = p$ and $\frac{d}{dt}\gamma_v(t)|_{t=0} = v$. The exponential map is then defined by

$$\exp_p : T_p\mathcal{M} \to \mathcal{M}, \quad v \mapsto \gamma_v(1)\,. \tag{67}$$

Intuitively, the exponential map formalizes the notion of taking a step in the direction of the tangent vector v and then finding the closest point on the manifold.

Gradient descent on manifolds: adding and scaling points on the manifold will not necessarily lead to points on the manifold. In contrast, such operations acting on tangent vectors do lead to other tangent vectors, since a tangent space is a vector space. This suggests a generalization of gradient descent on manifolds: we first scale the gradient of a loss $\mathcal{L} : \mathcal{M} \to \mathbb{R}$, which in particular is a tangent vector, by the learning rate $\lambda \in \mathbb{R}$. To obtain the updated point $p^{i+1} \in \mathcal{M}$, we apply the exponential map at the previous point $p^i \in \mathcal{M}$ to the rescaled gradient:

$$p_{i+1} \leftarrow \exp_{p_i}(-\lambda \operatorname{grad}^G_{p_i} \mathcal{L}) \,. \tag{68}$$

In $\mathbb{R}^n$, the exponential map is $\exp_p(v) = p + v$ leading to standard gradient descent.

Right multiplication: Lie groups are smooth manifolds endowed by an additional group multiplication. There is therefore a natural diffeomorphism given by right multiplication

$$R_g : G \to G \,, \qquad\qquad R_g : h \mapsto hg \,, \tag{69}$$

for $h, g \in G$. Let us denote the unit element of the group by e. Since right multiplication R_g is a diffeomorphism, its differential

$$(dR_g)_e : T_eG \to T_gG$$

is a linear isomorphism (linear bijective map). We can therefore uniquely identify tangent vectors $v \in T_gG$ and Lie algebra elements $\tilde{v} \in \mathfrak{g} \simeq T_eG$.

Matrix groups and right multiplication: For matrix groups, the differential $(dR_g)_e : T_eG \to T_gG$ takes a particularly simple form

$$(dR_g)_e v = vg \tag{70}$$

where we take the standard matrix product of the Lie algebra element v and the group element g on the right hand side. This statement can be easily checked by noticing that the differential acts on any function $f : G \to \mathbb{R}$ by

$$[(dR_g)_e v]f = \frac{d}{dt} f(\underbrace{\gamma_v(t)g}_{\equiv \tilde{\gamma}(t)})\Big|_{t=0} \,, \tag{71}$$

where γ_v denotes the curve associated with $v \in T_eG$, i.e. $\gamma_v(0) = e$ and $\frac{d}{dt}\gamma_v(t)|_{t=0} = v$. As a result, the curve $\tilde{\gamma}$ obeys

$$\tilde{\gamma}(0) = g \,, \qquad\qquad \frac{d}{dt}\tilde{\gamma}(t)|_{t=0} = \frac{d}{dt}\gamma(t)|_{t=0}\, g = vg \,, \tag{72}$$

and is therefore a curve associated with $vg \in T_gG$ as claimed.

Killing form: we first define the adjoint map

$$\operatorname{ad}_u(v) = [u, v] \tag{73}$$

where $u, v \in \mathfrak{g}$. The adjoint map is linear and a Lie algebra homomorphism

$$\operatorname{ad}_{[u,v]} = [\operatorname{ad}_u, \operatorname{ad}_v] \,. \tag{74}$$

Then the Killing form is given by

$$B(u, v) = \operatorname{Tr}(\operatorname{ad}_u \circ \operatorname{ad}_v) \tag{75}$$

for $u, v \in \mathfrak{g}$ is a bilinear symmetric form that is non-degenerate if the group is semi-simple and compact. We can use the Killing form to define a inner product for T_eG by

$$\langle u, v \rangle_e = -B(u, v) \,. \tag{76}$$

The negative sign ensures positive definiteness.

Riemannian metric: Often it is more convenient to work with the Lie algebra instead of the various tangent spaces. For example, an inner product $\langle \cdot, \cdot \rangle_e$ on the Lie algebra induces a Riemannian inner product on the group G by

$$\langle v, w \rangle_g \equiv \langle (dR_{g^{-1}})_g v, (dR_{g^{-1}})_g w \rangle_e = \langle vg^{-1}, wg^{-1} \rangle_e \,, \qquad \text{for } v, w \in T_g G \,, \tag{77}$$

where we have used that $dR_{g^{-1}} = (dR_g)^{-1}$ to pull tangent vectors back to the Lie algebra. Using $(dR_g)^{-1} = dR_{g^{-1}}$, it immediately follows that the inner product is right-invariant, i.e.,

$$\langle (dR_g)_e \xi, (dR_g)_e \eta \rangle_g = \langle \xi, \eta \rangle_e \tag{78}$$

for Lie algebra elements $\xi, \eta \in T_e G$. For $SO(3)$, we choose antisymmetric generators T^a that are orthonormal with respect to the inner product induced by the Killing form $\langle T^a, T^b \rangle = \frac{1}{2}\text{tr}(T^a)^\top T^b = \delta^{ab}$.

Riemannian gradient: Recall that the Riemannian gradient $\text{grad}_g f \in T_g G$ is the the unique element of the tangent space which obeys

$$\langle \text{grad}_g f, v \rangle_g = df_g \, v \tag{79}$$

for all $v \in T_g G$. Due to the isomorphism between the tangent space $T_g G$ and the Lie algebra $\mathfrak{g} \simeq T_e G$, the Riemannian gradient $\text{grad}_g f \in T_g G$ uniquely corresponds to the Lie algebra element

$$\nabla f \equiv (dR_{g^{-1}})_e \, \text{grad}_g f \in \mathfrak{g} \,. \tag{80}$$

It is often more convenient to work with this Lie algebra representative of the Riemannian gradient. We will now derive a simple expression for this Lie algebra gradient. By definition of the Riemannian gradient, it holds that

$$\langle \text{grad}_g f, (dR_g)_e v \rangle_g = df_g \left((dR_g)_e v \right) \,, \tag{81}$$

for any $v \in \mathfrak{g}$. We can invert equation 80 to obtain

$$\text{grad}_g f = (dR_g)_e \nabla f \tag{82}$$

Using this, we can rewrite equation 81 as follows

$$\langle (dR_g)_e \nabla f, (dR_g)_e v \rangle_g = \langle \nabla f, v \rangle_e \,, \tag{83}$$

where we have used the right-invariance of the metric equation 78. We thus conclude that

$$\langle \nabla f, v \rangle_e = df_g \left((dR_g)_e v \right) \tag{84}$$

By definition of the right hand side, it therefore holds for a matrix group that

$$\langle \nabla f, v \rangle_e = df_g(vg) = \frac{d}{dt} f(\exp(tv) g) \bigg|_{t=0} \,. \tag{85}$$

We can expand the Lie algebra representative in terms of generators $\nabla f = \sum_a T^a \nabla^a f$ whose components are given by

$$\nabla^a f = \langle \nabla f, T^a \rangle_e = \frac{d}{dt} f(\exp(tT^a) g) \bigg|_{t=0} \,, \tag{86}$$

where we have used equation 85 and assumed that the generators are chosen to be orthonormal, i.e., $\langle T^a, T^b \rangle_e = \delta^{ab}$. For brevity, we typically refer to the Lie algebra representative of the Riemannian gradient ∇f as simply the Riemannian gradient in the main part of the paper.

Exponential map: For a Lie group, the exponential map takes the form

$$\exp_g(v) = \exp((dR_{g^{-1}})_g v) \, g \tag{87}$$

for $g \in G$ and $v \in T_g G$ where $\exp : \mathfrak{g} \to G$ is the matrix exponential for matrix Lie groups. We note that, by the Lie algebra expression of the Riemannian gradient equation 82, this implies that gradient descent on a Lie group with learning rate $\lambda \in \mathbb{R}$ and loss $\mathcal{L} : G \to \mathbb{R}$ is given by

$$g^{i+1} = \exp(-\lambda \nabla \mathcal{L}) g^i \,. \tag{88}$$

This result is used extensively in the main part of the paper.

Differential: Right multiplication $R_g : G \to G$ induces a isomorphism between the tangent spaces T_gG and the Lie algebra $\mathfrak{g}$. Consider the differential $df_g : T_gG \to T_{f(g)}G$ of the map $f : G \to G$. Using the right multiplicative isomorphism equation 70, we can write any tangent space element in terms of a Lie algebra element

$$T_gG \ni \Omega G \leftrightarrow \omega \in \mathfrak{g} \tag{89}$$

Specifically, we can define a Lie algebra representative of the differential

$$\mathfrak{D}f_g \equiv d(R_{f(g)^{-1}} \circ f_g \circ R_g)_e : T_eG \simeq \mathfrak{g} \to T_eG \simeq \mathfrak{g} \tag{90}$$

Using the fact that the right isomorphism amounts to right multiplication equation 70, we can derive a explicit expression for this representative

$$\begin{aligned}
\mathfrak{D}f_g(\omega) &= df_g(\omega g) f(g)^\top && (91)\\
&= \frac{d}{dt} f(\exp(t\omega)g)\big|_{t=0} f(g)^\top && (92)\\
&= \frac{df}{dg}(\omega g) f(g)^\top && (93)\\
&= \sum_a \omega_a \frac{df}{dg}(T^a g) f(g)^\top && (94)\\
&= \sum_a \omega_a \frac{d}{dt} f(\exp(tT^a g)\big|_{t=0} f(g)^\top && (95)\\
&= \sum_a \omega_a \nabla^a f(g)\ f(g)^\top . && (96)
\end{aligned}$$

Since $\mathfrak{D}f_g(\omega) \in \mathfrak{g}$, we can expand it as $\mathfrak{D}f_g(\omega) = \sum_b T^b \mathfrak{D}^b f_g(\omega)$. Hence, it holds that

$$\mathfrak{D}^b f_g(\omega) = \sum_a \langle T^b, \nabla^a f(g)\ f(g)^\top \rangle\, \omega_a \equiv \sum_a \mathfrak{D}f_g^{ba}\, \omega_a\,, \tag{97}$$

with

$$\mathfrak{D}f_g^{ba} \equiv \langle T^b, \nabla^a f(g)\ f(g)^\top \rangle\,. \tag{98}$$

We can think of $\mathfrak{D}f_g^{ba}$ as being the matrix representation of the differential in the generator basis T^a.

Chain rule: Let's also consider a function $F : G \to \mathbb{R}$ and its composition $F \circ f : G \to \mathbb{R}$. Then it holds that

$$\nabla^a (F \circ f)_g = \sum_b \nabla^b F_{f(g)}\, \mathfrak{D}f_g^{ba}\,. \tag{99}$$

This follows directly from

$$F \circ f \circ R_g = F \circ R_g \circ R_{g^{-1}} \circ f \circ R_g\,. \tag{100}$$

Taking the differential of this and using the standard chain rule of the differential then gives

$$d(F \circ R_{f(g)} \circ R_{f(g)^{-1}} \circ f \circ R_g)_e = d(F \circ R_{f(g)})_e\, d(R_{f(g)^{-1}} \circ f \circ R_g)_e = \nabla F_{f(g)}\ \mathfrak{D}f_g\,. \tag{101}$$

E SO(3) Conventions

Any group element $g \in SO(3)$ can be written as

$$g = \exp(A) \tag{102}$$

with $A \in \mathfrak{so}(3)$ taking value in the Lie algebra

$$\mathfrak{so}(3) = \{A^\intercal = -A \,|\, A \in \mathbb{R}^{3,3}\}\,. \tag{103}$$

A Lie algebra is, in particular, a vector space. We will choose the basis T^a with $a \in \{1,2,3\}$ with

$$T^1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, \qquad T^2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}, \qquad T^3 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{104}$$

In the context of Lie theory, the basis vectors T^a are also referred to as the generators of the Lie algebra. In particular, any antisymmetric three-by-three matrix can be written as a linear combination of the generators. Furthermore, the generators obey the following commutation relations

$$[T^a, T^b] = \sum_{c=1}^{3} \epsilon^{abc} T^c \,, \tag{105}$$

where ϵ^{abc} denotes Levi-Civita symbol. It can be checked that

$$\mathrm{Tr}(T^a) = 0 \,, \tag{106}$$

$$\mathrm{Tr}(T^a T^b) = -2\delta^{ab} \,, \tag{107}$$

where δ^{ab} is the Kronecker symbol. Using equation 107, we can equip the Lie algebra with an inner product

$$\langle A, B \rangle = \frac{1}{2} \mathrm{Tr}(A^\intercal B) = \sum_{a=1}^{3} A^a B^a \,, \tag{108}$$

where we have used that we can express an arbitrary element A of the Lie algebra $\mathfrak{so}(3)$ as $A = \sum_{a=1}^{3} A^a T^a$. Note that our basis T^a is orthonormal with respect to this inner product.

F Limitations of Guidance

Guidance is a technique to bias the generation process, i.e., instead of sampling from the model distribution p, one aims to sample from

$$p_{\text{tilted}}(x) = p(x) \frac{1}{Z_{\text{guide}}} e^{-\beta E_{\text{guide}}(x)} \tag{109}$$

where E_{guide} denotes the guidance energy of interest. For score based models, this amounts to a modification of the score

$$\nabla_x \log p_{\text{tilted}}(x) = \nabla_x \log p(x) - \beta \nabla_x E_{\text{guide}}(x) \,. \tag{110}$$

A challenging aspect of guidance is diffusion involves a one-parameter family of density which is typically parameterized in terms of diffusion time t or noise level $\sigma(t)$. For many guidance energies, such as the Rosetta energy function, it is highly non-trivial to calculate the score of the guidance density at time t,

$$p_{\text{guide}}(x_t) = \frac{1}{Z_{\text{guide}}} \int p(x_0|x_t) e^{-\beta E_{\text{guide}}(x_0)} dx_0 \,, \tag{111}$$

as it involves an untractable marginalization over the unnoisy datasample x_0. Here, $p(x_0|x_t)$ denotes the backward kernel, i.e. the density of obtaining the unnoisy data sample x_0 when integrating the reverse SDE starting from x_t at diffusion time t. The fundamental challenge of all guidance schemes is to approximate the above marginalization in a suitable manner for the application at hand. This approximation becomes particularly challenging in the limit of $\beta \to \infty$ which corresponds to sampling minimizers of the guidance energy E_{guide}. In this limit, the integrand is heavily dominated by the minimizers (and its neighborhoods for very large but finite β as used in numerical experiments). As such, one has to estimate the conditional density $p(x^*|x_t)$ for the minimizers x^* with high precision. For this reason, guidance cannot be expected to work well for minimizing generic guidance energies.

Let us consider the simpler case in which we do not want to minimize the guidance energy, i.e., we do not consider the challenging $\beta \to \infty$ limit. In this case, there are various strategies to approximate the marginalization. None of them are perfect, and each comes with a certain set of tradeoffs. We will

only focus on methods which work for a pretrained unconditional model as this is the relevant setting for diffeomorphic optimization. In principle, one could use Monte-Carlo to estimate the guided score

$$\nabla_{x_t} \log p_{\text{guide}}(x_t) \approx \nabla_{x_t} \log \frac{1}{N} \sum_{i=1}^{N} e^{-\beta E_{\text{guide}}(x_0^{(i)}(x_t))} \tag{112}$$

with $x_0^{(i)} \sim p(x_0|x_t)$. For this, we have to repeatedly integrate the reverse SDE starting from x_t and backpropagate through it. There are methods for refined backpropagation through SDEs, such as suitable generalizations of the adjoint state methodKidger et al. (2021); Li et al. (2020) but we are not aware of any work in the protein space applying these techniques. This is possibly due to the considerable technical difficulty, such as the appearance of Stratonovich stochastic integrals, as well as scaling concerns to larger problem sizes. Furthermore, one would need to derive generalizations of these methods for matrix Lie groups.

Universal guidance Bansal et al. (2023) is a technique inspired by classifier guidance Dhariwal & Nichol (2021). In this setting the marginalization is approximated by a zeroth-order saddle point approximation around the expectation value $\hat{x}_0(x_t) = \mathbb{E}[x_0|x_t]$ of the kernel $p(x_0|x_t)$, i.e.,

$$p_{\text{guide}}(x_t) \approx \frac{1}{Z_{\text{guide}}} \int \delta(\hat{x}_0(x_t) - x_0)\, e^{-\beta E_{\text{guide}}(x_0)} dx_0 = \frac{1}{Z_{\text{guide}}} e^{-\beta E_{\text{guide}}(\hat{x}_0(x_t))}\,, \tag{113}$$

which can be a good approximation at low noise levels (small diffusion time) but tends to be very poor at high noise. To alleviate this problem, guidance is often combined with a heuristic schedule which switches the guidance score on when the diffusion time is lower than some threshold (often in an adiabatic manner). However, this results in a fundamental tension: we want the guidance signal to kick in as early as possible in the generation process to meaningfully change the selected mode of the sample. On the other hand, we do not want to start the guidance too early as the zeroth-order saddle point approximation would be very poor. Whether such a delicate balance can be struck often heavily depends on the problem setting and requires careful (and costly) hyperparameter tuning. As a general rule of thumb, energy functions that depend only on the coarse grain details of the sample x_0 seem to be easier to deal with. Examples include the classification of image types or the sentiment of a sentence. However, energy functions in physics (which are highly relevant in the protein space) are typically not of this type. For example, the Rosetta energy is extremely sensitive to the repulsive component of the atomic interactions resulting in numerically destabilizing energy values even if the global fold and secondary structure of the predicted unnoisy protein is roughly correct.

Recently, there has been considerable interest in combining diffusion models with Sequential Monte Carlo (SMC) to facilitate sampling from the tilted distribution p_{tilted}. We base our discussion on the very recent Singhal et al. (2025) which introduces Feynman-Kac steering unifying a number of previous approaches in a common theoretical framework. The basic idea is to modify the transition kernel $p(x_{t-1}|x_t)$ of the reverse diffusion

$$p(x_t|x_{t+1}) \to p(x_t|x_{t+1}) G(x_1, \ldots, x_t) \tag{114}$$

by a guidance potential G which fullfills the consistency condition

$$\prod_{t=1}^{0} G(x_1, \ldots, x_t) = \frac{1}{Z_{\text{guide}}} e^{-\beta E_{\text{guide}}(x_0)}\,. \tag{115}$$

This ensures that the joint distribution of the sampling trajectory $(x_1, \ldots, x_t, \ldots x_0)$ is given by

$$p_{\text{guide}}(x_1, \ldots, x_0) = p(x_1, \ldots, x_0) \frac{1}{Z_{\text{guide}}} e^{-\beta E_{\text{guide}}(x_0)} \tag{116}$$

and thus has the desired marginal $p_{\text{guide}}(x_0)$. In principle, any guidance potential satisfying the consistency condition will lead to asymptotic guarantees but in practice a careful choice of potential is vital to avoid prohibitive variance. This is because, at each step of the reverse diffusion one uses importance sampling to reweight the sample x_t according to the tilted transition kernel equation 114. In practice, one often has to choose guidance potentials G that involve challenging marginalization. For example, a widely used guidance potential is the expected guidance energy E_{guide}. Thus this approach faces analogous challenges to universal guidance, particularly in the $\beta \to \infty$ limit.

Another important point of differentiation between guidance and diffeomorphic optimization is that the latter is particularly suited in applications in which we want to improve a particular sample x_0 with respect to some cost function. For guidance, we would need to incorporate a constraint in the guidance potential that ensures similarity to x_0. For diffeomorphic optimization, we perform gradient descent updates that iteratively refine the sample.

G Complexity and Efficiency of Gradient Estimations through Flows

A forward pass evaluates the network layer by layer, applying linear transformations and nonlinearities to propagate an input through the model. Its cost is determined by the size and number of layers, with matrix multiplications dominating the total FLOPs. A general backward pass is more expensive because it must compute gradients with respect to all parameters. This involves propagating gradients backward through each layer and forming gradient tensors for every weight matrix in each layer and the layers input for further backward propagation. These extra operations roughly double the compute relative to the forward pass, and they also require storing all intermediate activations, which increases memory consumption and can become a limiting factor on modern hardware.

A backward pass that computes gradients only with respect to the input is cheaper. It is comparable to a single backward pass as the parameter gradients do not have to be computed. The computational effort becomes close to that of the forward pass, with only a small overhead for the backward sweep.

Gradient checkpointing reduces memory consumption during backpropagation by storing only a subset of intermediate activations from the forward pass and recomputing the missing ones when needed. In standard backpropagation, all activations must be kept in memory because each layer's gradient depends on its forward output; this creates a memory footprint that grows linearly with depth and often becomes the bottleneck in training large models. Gradient checkpointing breaks the computation graph into segments, saves only the boundary activations, and discards the rest. During the backward pass, the discarded activations are recomputed on the fly by running partial forward passes within each segment. This trades additional compute for a substantial reduction in memory usage. The memory cost can drop from linear to roughly the square root of the number of layers in optimal schemes, enabling deeper networks or larger batch sizes on the same hardware. The extra compute cost is typically a factor of 1.5–2×, but the trade-off is advantageous when memory is the limiting resource. Modern implementations generalize this idea through customizable checkpoint policies, selective recomputation, and integration with automatic differentiation frameworks, making gradient checkpointing a standard technique in large-scale training, especially for transformer architectures and diffusion models where activation tensors dominate memory load.

ODE solvers compute trajectories by applying numerical integration steps, with cost driven by repeated evaluations of the vector field. Higher-order or adaptive methods improve accuracy but increase per-step compute, so total complexity depends on both the solver order and the number of required steps. In neural ODEs, most of the expense comes from evaluating the neural network that defines the dynamics. Using the autograd engine to compute gradients through a ODE solution keeps the entire computational graph in memory, resulting in $\mathcal{O}(NL)$ scaling with a second pass over the graph in reverse direction required for the backward pass with a total cost of $\mathcal{O}(NL)$.

The adjoint method recomputes the state trajectoy and backpropagates the gradients by integrating a separate reverse-time ODE, avoiding storage of intermediate states and keeping memory use low. Compared to gradient checkpointing it does not require storing any values besides the vectorfield as it recomputes both the trajectory and the gradients during the backward pass. This leads to a memory complexity of $\mathcal{O}(L)$ as only the network parameters and their activations need to kept in memory at any time. This comes at the price of additional solver calls and potential numerical instability or even divergence from the trajetory of the forward pass, since the backward integration can amplify errors and often requires tighter tolerances. After $\mathcal{O}(NL)$ steps of compute in the forward pass, the adjoint method recomputes the forward pass to reconstruct the trajectory as well as one backward pass to compute the gradient, resulting in an additional $\mathcal{O}(2NL)$ compute complexity during the backward pass. As a result, adjoint-based gradients are memory-efficient $\mathcal{O}(L)$ but computationally more expensive with $\mathcal{O}(3NL)$. Solving a reverse time ODE comes with potential numerical issues as the recomputed trajectories diverge and subsequently lead to diverging gradients.

Checkpointing provides a middle ground by storing selected solver states and recomputing segments during backpropagation. This reduces memory usage to $S = N/K$ at any given time where S is the segment length. Storing K checkpoints in the form of a state of the ODE is negligible as it amounts to K samples. For K checkpoints, we only have to solve S steps at any one time, while the recomputation between the K checkpoints amounts to going forward and backward on the segments compute complexity of $\mathcal{O}(2KSL) = \mathcal{O}(2NL)$. It balances compute and memory more predictably, making it a practical choice for many neural ODE applications.

To quantitatively measure the performance of both the gradient checkpointing and adjoint method, we evaluated each gradient against the ground truth gradient obtained with numerical differentiation. Figure 9 shows the gradient error over increasing modelling dimensions as a funtion of the step size and the checkpointing interval. For the Euclidean part, we integrated the function $dx_t = \cos(t/2\pi) \cdot x_t$ and for the $SO(3)$ part, we integrated $dR_t = \cos(t/2\pi) \sum_a^3 v_a T^a \theta(v)$ with $v = [1.0, -0.2, 0.1]^T$ and $\theta(v) = \|v\|$. The initial condition was sampled from a Gaussian distribution in Euclidean space and the respective axis-angle space for the $SO(3)$ elements. The dimensionality was determined by the number of $SE(3)$ ODE's integrated in parallel for $t \in [0, 1]$. We used this toy ODE's to establish the numerical behavior of the autograd checkpointing and the adjoint methods for backpropagating gradients through ODE solvers.

The analysis suggests that the adjoint method with a checkpointing at every step yields gradients very close to its autograd gradient. Decreasing the checkpointing frequency requires recomputing longer parts of the trajectory which accumulates errors. Overall a trend emerges that decreasing the step size is beneficial for both adjoint and autograd gradients, and particularly for very small stepsizes, the difference reduces significantely.

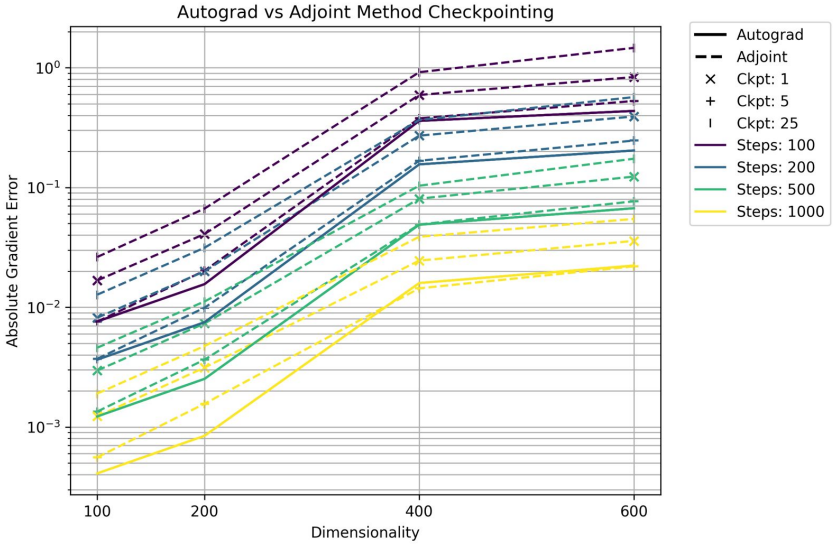


Figure 9: Absolute gradient error of autograd (solid) and the adjoint method (dashed) for the toy $SE(3)$ ODE as a function of dimensionality, i.e. the number of $SE(3)$ ODEs integrated in parallel. Colors denote the number of integration steps and markers the checkpointing interval of the adjoint method.

We also extended this analysis to the FrameFlow model Yim et al. (2023a) model (Figure 10) but due to the computational cost of numerical differentiation for the ground truth gradient refrained from evaluating multiple checkpointing frequencies. For this reason we kept the checkpointing frequency fixed at every five steps. One can observe that the adjoint gradients backpropagated through the $SE(3)$ ODE generally more imprecise than their autograd counterparts. For commonly used stepsizes of $1/25$ to $1/100$, the total deviation can be quite significant, but correcting for the per dimension gradient deviation, the gradient error is small.

Finally, we compare the gradient error of the AlphaFlow architecture Jing et al. (2024) in Figure 11. Again, adjoint gradient estimators perform worse than their autograd counterparts. It should be

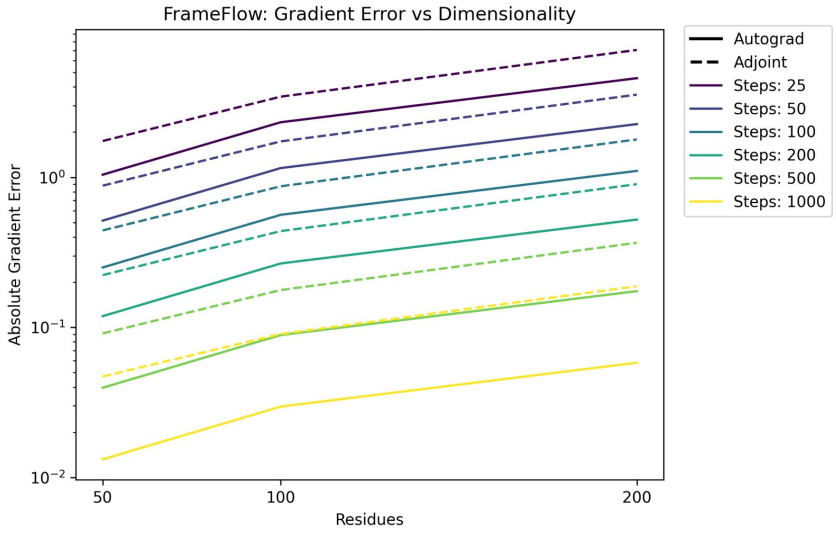


Figure 10: Absolute gradient error of autograd (solid) and the adjoint method (dashed) through the $SE(3)$ ODE of FrameFlow as a function of the number of residues. Colors denote the number of integration steps. The checkpointing interval is fixed to every five steps.

noted that AlphaFlow is a flow defined on the C_β carbon atoms and does not model a rotation group, compared to the previous two models. We want to note that the evaluation of these gradients were on the order of hours, particularly for the highest step sizes.

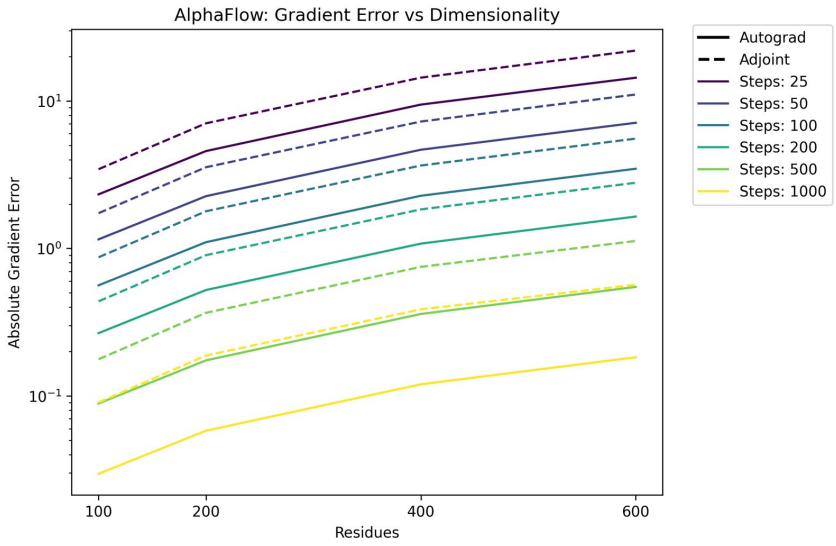


Figure 11: Absolute gradient error of autograd (solid) and the adjoint method (dashed) through the Euclidean ODE of AlphaFlow as a function of the number of residues. Colors denote the number of integration steps.

H EXPERIMENTS

H.1 SO(3) MANIFOLDS

We generate a manifold in the space of $SO(3)$ matrices. To plot the rotation matrices easily, we parameterize the SO(3) elements in their axis-angle vectors. The manifold in the axis-angle representation was generated with the S-curve function of sklearn, `sklearn.datasets.make_s_curve(n_samples=250000, noise=0.1)`.

The flow matching model was parameterized with a four layer MLP with SELU activation functions and 64 hidden units in each of the hidden layers. For training, the model was optimized with Adam using a learning rate of $1e^{-3}$ over 20 epochs on the training data with conditional flow matching to enforce that the predicted flow on the manifold of SO(3) matrices closely matches the target logarithmic map on the manifold. During sampling, the vector field was integrated with the Crouch–Grossman integrator with a step size of $dt = 1/200$ from 0 to 1.

For the sample optimization, a target sample z^* was experimentally determined and a second random sample was drawn as the initial value $x^{(0)} = g(z^{(0)})$ of the optimization problem. The mean squared error (MSE) between the target sample x^* and the sample $x^{(0)}$ being optimized served as the loss function $\mathcal{L}$. The sample $z^{(i)}$ was then optimized by calculating the gradient $d\mathcal{L}(g(z^{(i)}))/dz^{(i)}$ and performing 20 steps of gradient descent in the $SO(3)$ base space with a base space learning rate of 0.1.

The tangent space was calculated by evaluating the Jacobian matrix J_g with `torch.nn.functional.jacobian` at point z and its corresponding target space point $x = g(z)$. Subsequently, the two largest eigenvalues with the corresponding left columns were computed with a SVD and plotted with a scaling of 1.5 the point x.

H.2 SECONDARY STRUCTURE OPTIMIZATION

The secondary structure experiments were conducted on top of the FrameFlow codebase Yim et al. (2023a). For training and architectural details, we refer to the manuscript.

We used the model parameters in the provided model checkpoints by the original authors. The samples x with the time index $t \in [0, 1]$ consist of backbone elements of each residue centered on the Ca carbon atom. Each of the L backbone elements is parameterized by its translation and rotation and are elements of the SE(3) group. The flow generates an element (t, R) of the $SE(3)$ group for each residue. Upon completion of the sampling process, these transformations are applied to the idealized backbone coordinates.

For the experiment described in the main part of the paper, the loss function $\mathcal{L}$ was calculated as a function of the distance between the atom positions in the first and the last residue. The distance loss was implemented by taking the ReLU function over the difference between the desirable target distance and the actual distance, i.e. $\mathcal{L} = \left(\max(0, d^* - d(x_0^{(i)}, x_L^{(i)}))\right)^2$ where the target distance d^* between the atom positions in the two termini $x_0^{(i)}$ and $x_L^{(i)}$ in the backbone structure of length L at optimization step (i) of the structure was chosen as $d^* = L/2$. The distance metric was chosen as the mean squared error $d(x, y) = \sqrt{\sum_j (x_j - y_j)^2}$ over the atoms j in two residues x and y. The order of subtraction $\max(0, d^* - d)$ ensures a loss that becomes zero if $d \geq d^*$ and that the two termini are repelled from each other. We also experimented with an attraction loss in the sense of $\max(0, d - d^*)$ which worked equaly well. In fact, we were able to dynamically switch losses during optimization and the termini were pulled apart and together smoothly as determined by the loss with all intermediate optimization states $x^{(i)}$ staying on the data manifold. When switching between attracting and repelling loss function it was beneficial to reset the momenta of the adaptive optimizers like Adam to accelerate the optimization. For sampling we integrated with the Euler integration method with a step size of $dt = 1/25$. The learning rate for the optimization was chosen at a constant 0.5 for 300 optimization steps.

For the optimization of the dihedral angles according to the ABEGO scheme, we utilized a differentiable implementation of Rosetta energy function in PyTorch, i.e. tmol Leaver-Fay et al. (2025). We

use a specific component of the full energy function that takes the dihedral angles of neighboring backbone elements as an input. We modify this component such that the region of the Ramachandran plot which does not correspond to the desired secondary structure is penalized, which biases the optimization towards the chosen target region. We describe the construction below for the case in which the A region is the target, and the experiments targeting the B region use the same construction with the roles of the two regions exchanged.

In more detail, the combined Ramachandran/p_aa_pp energy term in tmol interpolates the energies (the negative-log probabilities) stored in a toroidal-grid lookup-table over phi and psi dihedrals. To push the generated samples away from the β-sheet region of conformation space, we created our own custom lookup table where we dramatically increased the energies for an elliptical region of the Ramachandran map that covers the β-sheet region. The energies at the edge of this ellipse are not uniform, so our first crude assignment of energies inside this ellipse that sloped away from a central peak to a particular value at its edge frustrated the minimizer, trapping conformations in the weird minima at the edge of the ellipse. In our second attempt, we instead labeled each grid cell inside the ellipse by the closest cell outside of the ellipse and then interpolated the energies along the distance from the center of the ellipse (to which we assigned an energy of +20) to the ellipse's edge. This led to the desired behaviour and we therefore chose this approach. The resulting grid is shown in Figure 12. This is the grid used for the results in Figure 5 and Table 2, for which a higher A percentage is therefore the desired outcome.

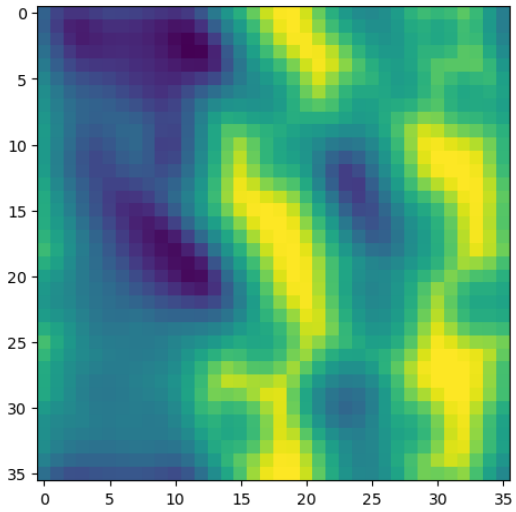


Figure 12: Modified Ramachandran energy grid used as the optimization objective, in which the β-sheet region is penalized so that the optimization is biased towards α-helices.

Three important metrics in machine learning aided protein design are *designability*, *diversity* and *novelty*. We consider designability the most pertinent to our task as diversity and novelty are of lesser interest when trying to converge on a single optimal structure. The optimization towards a single structure makes diversity collapse as expected. Similarly, since our aim is not to generate diverse samples but rather optimize towards a singular structure, novelty is a misplaced metric in this case. Empirically, the designability of the optimization trajectories agree well with the designability of the initial values drawn from the generative model. Only for larger learning rates of 1.0 does the designability experience a significant drop while at the same time only marginally improving the percentage of dihedral angles classified as B in the ABEGO scheme.

Since the FrameFlow model does not predict side chains, we interpreted all residues as alanine. We applied the sampled translations and rotations to the idealized backbone atom14 positions to obtain the sampled backbone geometry. The score function was minimized with gradient descent in the base space with a learning rate of 0.1 for 500 steps.

We implemented several versions of guidance to obtain maximally competitive baseline results. Specifically, we implemented both a single step and and a multi step denoiser combined with the loss function described above. The guidance was started at $t = 0.1$. We selected the starting

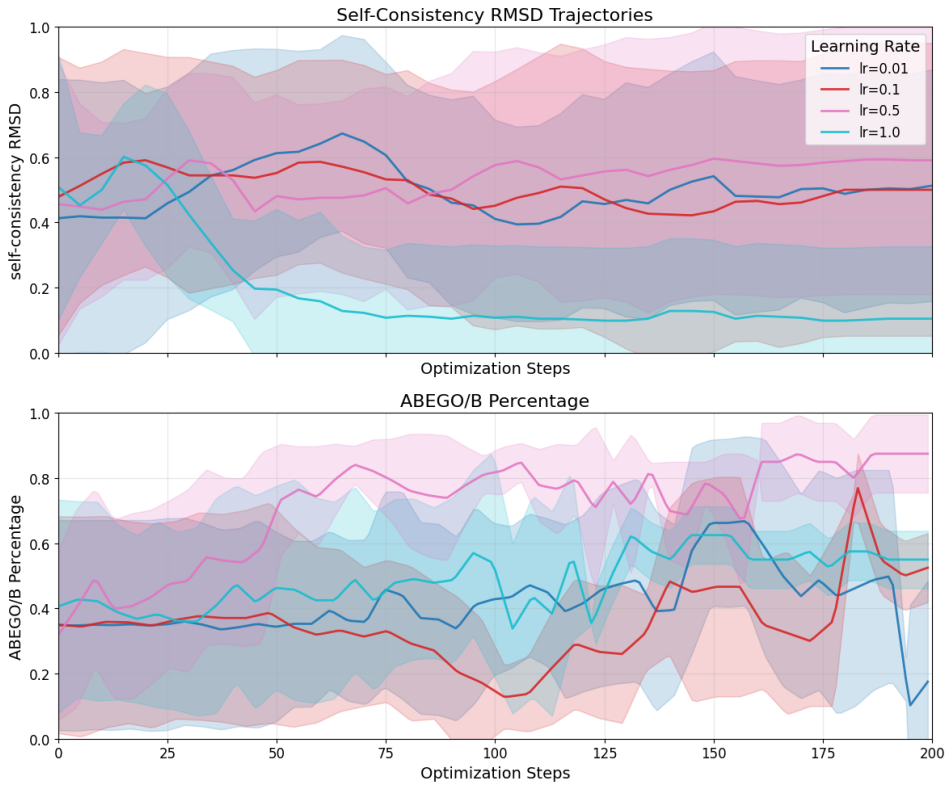


Figure 13: Fraction of designable samples (scRMSD $<$ 2 Angstroms, higher is better) and the optimized ABEGO percentage over the course of diffeomorphic optimization. Diffeomorphic optimization maintains a high fraction of designable samples throughout the optimization process of the secondary structure.

point of the guidance signal by hyperparameter line search over $t = 0.8, 0.1, 0.25, 0.5, 0.75$. We also found that it was benefical to stop sampling early at $t = 0.98$ to prevent distortive numerical pathologies that arose due to the divisor of the optimal transport transport path close to $t = 1$. The guidance vector field was obtained by evaluating the loss on the atom14 positions of each backbone residue and computing the gradient with respect to the corresponding element in the base space. For weighting the guidance term, we tested the theoretically motivated weighting function b_t from Lipman et al. (2024) but obtained better results with a constant, time-independent weight of $w = 25$. This value was selected using a line-search hyperparameter sweep for $w = 0.1, 1, 10, 25, 50, 100, 1000$. For evaluating the guidance term on a sample x_t, we tested both taking the direct gradient of the loss function, $\partial L(x)|_{x=g(x_t)}/\partial x$, and backpropagating back through the denoiser $\partial L(g(x_t))/\partial x_t$. We chose the latter as it yielded better results. Experimentally, we also found that only using the translation term (and not the rotational part) as guidance improved the performance. This may be due to the fact that on $SO(3)$, there is no equivalence between score and denoising as in Euclidean space. We furthermore carefuly ablated the number of denoiser steps used in multistep denoising. This is because fair comparison to diffeomorphic optimization should allow for a generous budget of denoising steps as the method is numerically costly. As shown in Table 2, even significant increase in the denoising steps does not lead to a pronounced gain in performance.

To demonstrate that the numerical cost of diffeomorphic gradient calculations can be reduced by coarse-grained backpropagation through the solver, we performed the following ablation study: we measured the performance of diffeomorphic optimization as a function of the number of steps used for sampling and calculating gradients during optimization. To this end, we ran diffeomorphic optimization with $10, 25, 50, 75$ and 100 integration steps during optimization, and evaluated the optimized sample after optimization with 1000 integration steps. The results are shown in Figure 14. Compared to guidance, even a very small number of steps used for sampling during optimization yields good performance. For this experiment, we measured the performance over 10 seeds as compared to 50 seeds in the main part.

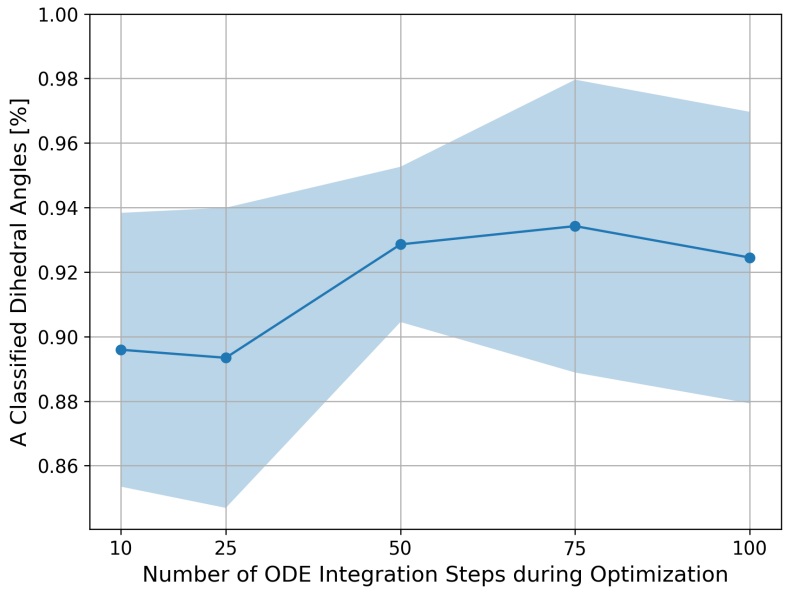


Figure 14: The performance of diffeomorphic optimization (higher is better) as a function of the number of ODE integration steps taken for each evaluation during optimization. After optimization, we evaluated all optimized samples with a 1000 steps. Diffeomorphic optimization is robust even when using just 10 integration steps during optimization.

H.3 Docking Optimization

For docking optimization we used the DiffDock code base. This diffusion model samples from the manifold of permissible rotational degrees of freedom of a small molecule on $SO(2)$ and its

			Secondary Structure A	Secondary Structure B
Unguided			53.5 ± 23.3	32.8 ± 20.1
	ODE Steps	Denoiser Steps		
Guidance	200	1	63.3 ± 25.1	27.7 ± 23.3
		50	63.5 ± 24.8	27.4 ± 23.1
		100	63.8 ± 24.5	27.1 ± 22.9
		200	64.1 ± 24.1	26.8 ± 22.6
	500	1	64.2 ± 24.4	26.6 ± 22.4
		50	64.4 ± 24.0	26.3 ± 22.1
		100	64.7 ± 23.7	26.0 ± 21.9
		200	65.0 ± 23.4	25.8 ± 21.6
	1000	1	65.0 ± 23.6	25.5 ± 21.5
		50	65.1 ± 23.4	25.3 ± 21.3
		100	65.2 ± 23.2	25.1 ± 21.0
		200	65.3 ± 23.1	24.9 ± 20.8
	ODE Steps	Optim Steps		
Diffeomorphic	25	500	$\mathbf{91.3 \pm 8.4}$	$\mathbf{7.4 \pm 7.3}$

Table 2: For this comparison the Ramachandran energy is biased towards the A region corresponding to α-helices, so that a higher A percentage indicates a better result. The addition of more compute for guidance leads to diminishing returns. Increasing the the number of integration steps both in the flow sampling as well as in the denoiser does not automatically lead to a higher target metric as measured with the percentage of A classified dihedral angles. All guidance terms were weighted with a constant $w = 25$. Diffeomorphic optimization samples with 25 integration steps and applies 500 gradient descent steps in the base space. Diffeomorphic optimization yields higher A classification scores with a substantially lower standard deviation in performance.

global translation and rotation on $SE(3)$. In order to make the sampling process differentiable, we implemented the probability flow formulation of the diffusion models which provided a deterministic map between base space and target space. For a differentiable loss, we considered two different metrics: the confidence head of the diffusion model and the OpenDock scoring function Hu et al. (2024).

Opendock is a physics-inspired scoring function and implements the Vina Score in a differentiable manner in PyTorch. One of the technical challenges of incorporating OpenDock as a suitable loss function were posed by its reliance on PDBQT file formats. In addition to the positional information stored in the PDB file format, the PDBQT file format additionally stores the partial charges, torsional bonds, and autodock atom types. To work around this constraint, we first saved the ligand conformations generated by DiffDock to disk in the SDF file format, converted it to the PDBQT and reloaded it with the additionally generated partial charges created during the conversion. At this point the computational graph connecting the base space variables to inputs of the differentiable loss scoring function was suspended due to the reading and writing to disk. To reconnect the computational graph with the input of the Opendock scoring function, we substituted the atom positions in the loaded PDBQT data structure with the atom positions originating from the DiffDock model.

The confidence head of the diffusion model is a neural network and was trained to predict whether the generated conformation is within 2 Angstroms of the ground truth. We additionally include on the scoring of OpenDock into our experimental setup which is a differentiable implementation of Vina. Following the original authors recommendation on the confidence prediction, we screened for the entries in the publicly available PDBBind test data set that were assigned a confidence logit of larger than 0 by the DiffDock confidence head. This resulted on a subset of roughly 78 PDBBind ligand-protein pairs on which the results were reported. We perform 10 optimization steps with an Adam-style optimizer with a learning rate of 0.1. Experimentally, we observe that the gradients of the rotation and torsional degrees of freedom play a dominant role in the norm of the total gradient.

To counteract this, we introduced separate scalings for the learning rates of the rotation and torsions and multiply the learning rates for these two sets of degrees of freedom by an additional 0.1.

To ensure fair comparison, we draw more samples for the sampling baseline in order to match the computational budget used by diffeomorphic optimization. Since both checkpointing and the adjoint method recompute the activations during the backward pass through the ODE solver, we used a budget corrector of $3\times$ a single sampling step. This budget corrector number could be reduced more with coarser checkpointing and fewer steps, but we observed favorable experimental results even for this setting and thus did not explore this direction further.

Figure 15 compares, for each protein-ligand pair, the best OpenDock Top-1 score reached by diffeomorphic optimization against the best score of the budget-matched sampling baseline. All pairs lie on or below the diagonal, i.e. diffeomorphic optimization never returned a worse Top-1 score than iid sampling at the same computational budget.

Since the OpenDock score is also the quantity that we optimize, we additionally verify that the improved scores correspond to physically sensible poses. Figure 16 reports the ligand RMSD to the crystal structure before and after optimization. The fraction of poses within 2 Angstroms of the crystal structure increases from 79.5% to 84.6% and the median change in RMSD is -0.03 Angstroms. The improvement in the docking score is thus not obtained by displacing the ligand away from its experimentally determined pose. The optimization is nevertheless not uniformly beneficial: 9 of the 78 pairs degrade by more than 0.25 Angstroms and one of the 62 initially sub-2-Angstrom poses loses this status.

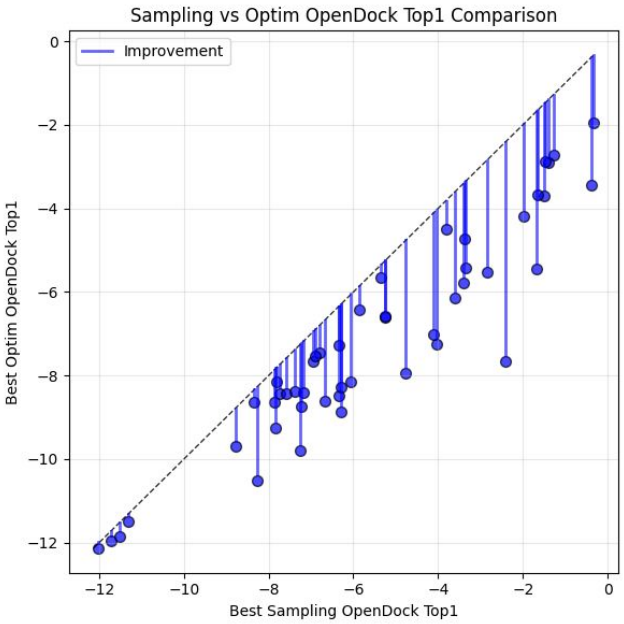


Figure 15: Per-pair comparison of the best OpenDock Top-1 score obtained by budget-matched iid sampling (horizontal axis) and by diffeomorphic optimization (vertical axis) on the 78 PDBBind test pairs. Lower scores are better and the vertical segments indicate the improvement obtained for each pair. Every pair lies on or below the diagonal, i.e. diffeomorphic optimization never returns a worse Top-1 score than the sampling baseline at the same computational budget.

H.4 Tertiary Structure Optimization

We built on the alphaflow codebaseJing et al. (2024) and chose the ESMFold-based architecture as it does not require costly MSA. We ensured that diffeomorphic optimization works both for distilled and non-distilled case but large-scale test-set-wide experiments were conducted with the latter as it leads to significant speed-up. The model ingests the pairwise distances between random atom positions to predict the backbone and the side chain atom positions conditioned on the sequence information of the

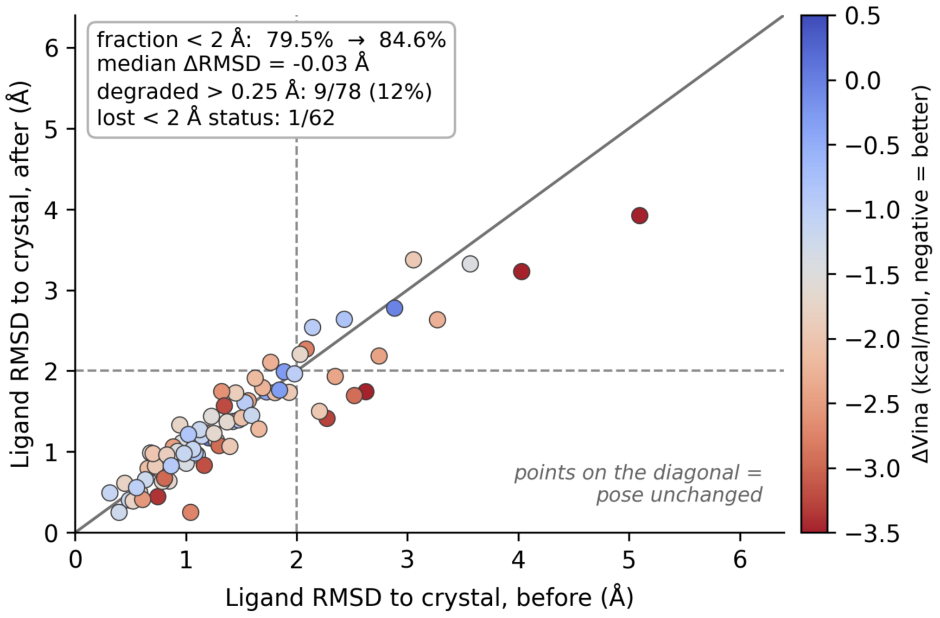


Figure 16: Ligand RMSD to the crystal structure before and after diffeomorphic optimization of the OpenDock score, colored by the corresponding change in Vina score where negative values indicate an improvement. Points on the diagonal correspond to poses that are left unchanged. The fraction of poses within 2 Angstroms of the crystal structure increases from 79.5% to 84.6% at a median RMSD change of -0.03 Angstroms, which shows that the gain in the docking score is not achieved by moving the ligand away from its experimentally determined pose. At the same time, 9 of the 78 pairs degrade by more than 0.25 Angstroms and one of the 62 initially sub-2-Angstrom poses loses this status.

protein. As a differentiable loss function, we used the tmol implementation of of beta_nov2016_cart Rosetta score function which support pytorch autograd to compute gradients. As the architecture contains non-differentiable distograms, we implemented a differentiable straight-through version thereof harnessing the sigmoid function σ,

$$d_{soft} = \sigma(\beta \cdot (d - lower)) \cdot (1 - \sigma(\beta \cdot (d - upper))$$
$$d_{gram} = \text{detach_grad}(d) + (d_{soft} - \text{detach_grad}(d_{soft}))$$

where $lower$ and $upper$ denote the lower and upper bound of the discrete bin in which the distance d should have been placed. While the forward pass is unaffected by the soft distogram implementation, the gradients are affected by the smooth approximation. Higher values of the β temperature result in a steeper sigmoid function at the cost of a stiffer gradient surface.

As a baseline we used the state-of-the-art Rosetta Relax protocol Alford et al. (2017) without any diffeomorphic optimization. For random sidechain repacking, Rosetta Relax uses n seeds which are commonly referred to as structures in the Rosetta community. This is followed by gradient descent in which the repulsive terms in the energy function are gradually increased with a linear schedule. Packing and gradient-based minimization is then repeated k times.

In Figure 19 we evaluated the Rosetta Relax baseline with an increasing amount of compute measured as the product of the number of seeds $1, 3, 5$ with the number relaxation cycles $3, 5$. We were not able to discern an improvement compared to diffeomorphic optimization in conjunction with Rosetta Relax.

For diffeomorphic optimization, we used Adam with a learning rate of 0.1 and otherwise standard pytorch hyperparameters. This was then followed by standard Rosetta relax to reduce numerical cost. The performance of this protocol as a function of diffeomorphic optimization steps is shown in Figure 20.

In addition to the experiments reported in the main part of the paper, we also compared the performance of our diffeomorphic Rosetta Relax to a sampling-based baseline. For this baseline, we simply sample from AlphaFlow with the equivalent computational budget used by diffeomorphic optimization. We select the sample with the lowest energy which is then minimized by the same standard Rosetta relax as used in the last step of the diffeomorphic protocol. As shown in Figure 18, diffeomorphic optimization again outperforms this baseline.

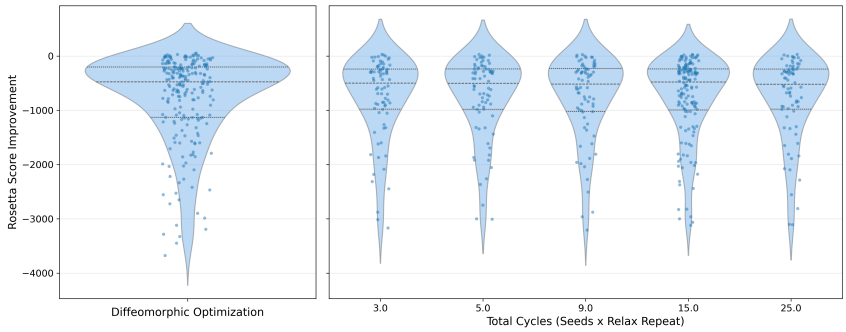


Figure 17: Diffeomorphic optimization of the Rosetta energy function: each point denotes a prediction of ESMFlow for an element of the pdb test set of Alphaflow for which we measure the improvement $E_{\text{diffeo}} - E_{\text{baseline}}$ of diffeomorphic Rosetta Relax over standard Rosetta relax. **Left:** Diffeomorphic Rosetta Relax outperforms Rosetta Relax and improves the Rosetta score with the default Rosetta Relax values of three structures and three repeated relaxations. **Right:** This improvement holds across an increasing computational budget as measured in terms of the total number of relaxations.

H.5 Lattice Gauge Theory

The protein experiments lack an exact criterion for whether an optimized sample still belongs to the data distribution. Two-dimensional lattice gauge theories provide such criteria in closed form, which is why we use them to probe diffeomorphic optimization directly.

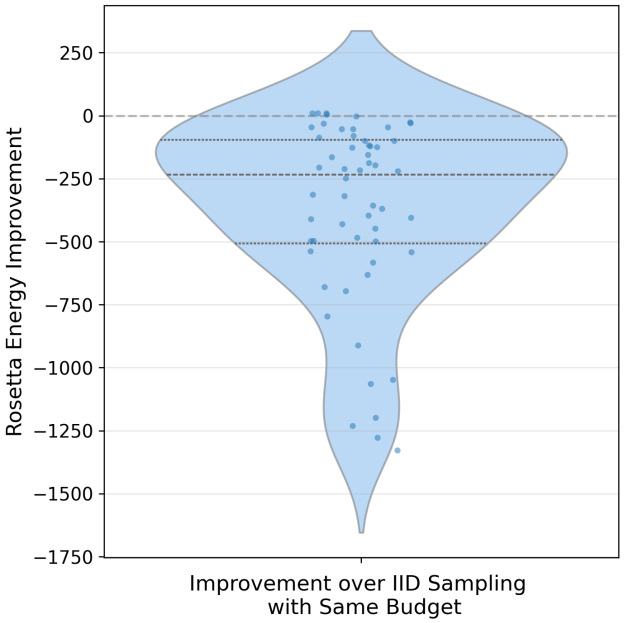


Figure 18: Diffeomorphic relax leads to lower Rosetta energy when compared to a sampling-based baseline. This baseline first samples structures from AlphaFlow using the same computational budget as used for diffeomorphic optimization. Then, standard Rosetta relax is used to minimize the energy of the sample with the lowest energy as in the diffeomorphic protocol. We plot the improvement of diffeomorphic optimization over this baseline, i.e., $E_{\text{diffeo}} - E_{\text{baseline}}$. Negative number indicates that the diffeomorphic protocol leads to lower energy values, as desired.

Factorization structure with $\mathbf{SU(3)}$: a configuration x assigns an $SU(3)$ matrix to every link of an 8×8 periodic lattice. A plaquette P is the product of the four matrices around an elementary square and $\langle P \rangle$ denotes the real part of its normalized trace $\operatorname{Re}\operatorname{tr} P/3$, averaged over the N_p plaquettes of the lattice. The Wilson action is $S = -\beta \sum_p \operatorname{Re}\operatorname{tr} P_p/3$ up to a constant, so that the mean plaquette is a direct measure of the loss level. The training data are Boltzmann samples $p(x) \propto e^{-S(x)}$ at $\beta = 8$ drawn with Hamiltonian Monte Carlo, and g is the probability flow ODE of a diffusion model trained on them. We take the cost $\mathcal{L} = S$ itself. Direct gradient descent on the links is a discretization of the Wilson flow, the standard smoothing procedure of lattice field theory, which is not designed to preserve the ensemble. Diffeomorphic optimization applies the gradient update of Section 4 to the base point, with the Lie group adjoint specialized to $SU(3)$.

Lowering the action moves a sample towards the Boltzmann ensembles of larger β, a one-parameter family whose position can be read off from $\langle P \rangle$. The descent takes the samples from $\beta = 8$ to an effective β of 33 to 37, which lies well outside the training distribution of g. In two dimensions the plaquettes of every member of this family are independent random matrices, so that the 2×2 Wilson loop $W_{2\times2}$ factorizes into its four plaquettes and the ratio $R_{2\times2} = \langle W_{2\times2} \rangle / \langle P \rangle^4$ equals one at every β up to negligible finite-volume corrections. This provides a parameter-free test of whether a descent leaves the family. Diffeomorphic optimization stays within 11% of one along the whole descent and ends at 0.95, which is within the resolution of the generative model itself whose unoptimized samples give $R_{2\times2} = 0.967 \pm 0.015$. Direct gradient descent breaks the identity to $R_{2\times2} = 0.59$ already at $\langle P \rangle = 0.82$. Neither heavy-ball momentum, a Fourier-space preconditioner nor guidance changes this result.

Topological sectors with $\mathbf{U(1)}$: Figure 1 motivates base space optimization with the ability to move between regions of the data distribution that a descent in the data space connects only through high-cost configurations. Compact $U(1)$ gauge theory in two dimensions makes such moves exactly countable. A configuration assigns an angle to every link of a 32×32 lattice, the plaquette angle θ_p is the sum of the four link angles around a square reduced to $(-\pi, \pi]$, and the topological charge $Q = \sum_p \theta_p/(2\pi)$ is an integer that labels the sector of a configuration. A continuous path can change

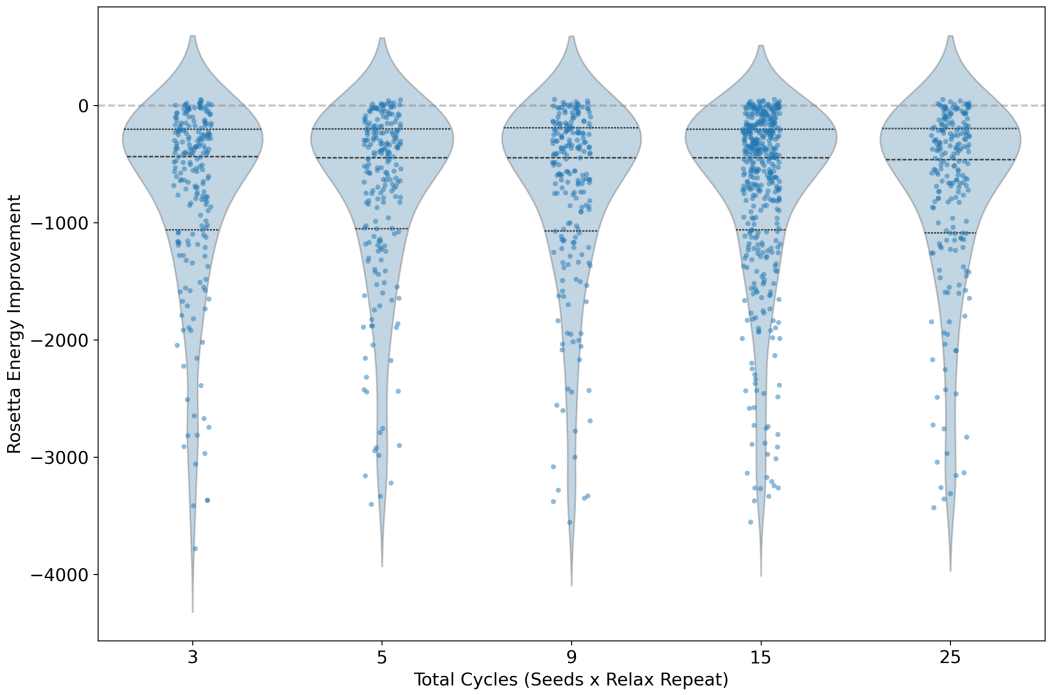


Figure 19: Diffeomorphic optimization in combination with Rosetta Relax yields improved (lower) Rosetta energies when directly compared to Rosetta Relax. This statement holds even with increasing computational budget for the baseline. In this experiment, we varied the number of Rosetta Relax random seeds (referred to as structures in the Rosetta community) and the number of relaxations for each random seed. The product of both Rosetta Relax random seeds and relaxations is plotted on the horizontal axis. We can observe no substantial benefits of providing Rosetta Relax with additional compute budget compared to diffeomorphic optimization which consistently improves the final Rosetta energy.

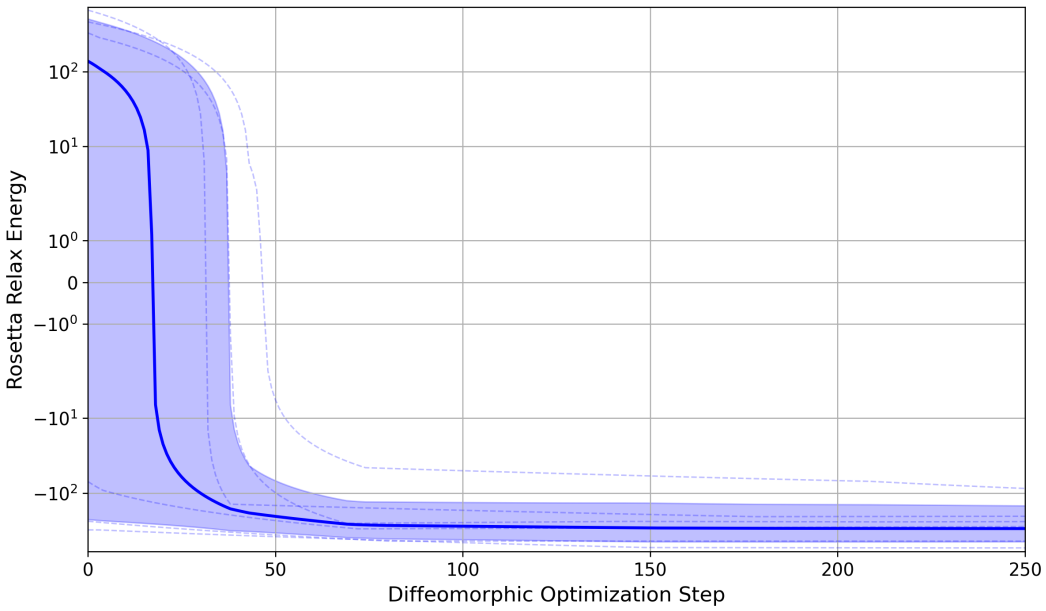


Figure 20: Diffeomorphic Optimization in combination with Rosetta Relax can achieve substantial energy reductions in a small number of steps. This is exemplified by the high energies at the beginning of the optimization curves which corresponds to running Rosetta Relax without a diffeomorphic optimization step which are subsequently is minimized by several orders of magnitudes while remaining stable.

Q only by driving one plaquette angle through $\pm\pi$, where its cost $1 - \cos\theta_p$ is maximal. We call a plaquette with $|\theta_p| > \pi/2$ a vortex. We train g on Boltzmann samples at $\beta = 4$ and take

$$\mathcal{L} = S + \mu\,(Q_f - Q_0 - 1)^2\,, \qquad \text{with } Q_f = \sum_p \sin(\theta_p)/(2\pi)\,, \tag{117}$$

where the smooth surrogate Q_f asks each configuration to move one sector up from its initial charge Q_0 and $\mu = 120$. Both methods start from the same Monte Carlo configurations, 256 per seed and three seeds, with diffeomorphic optimization entering the base space through the inverse of g. Both stop at the first iteration whose batch mean $\langle P\rangle$ reaches 0.95, the loss level of a Boltzmann ensemble at $\beta = 10$, at which such an ensemble carries 0.008 vortices per configuration. The stopping level matters: as $\langle P\rangle \to 1$ the plaquette angles become uniform, each sector contains a single such configuration, every method converges to it, and all diagnostics return to their exact values for free.

Diffeomorphic optimization places 98% of the configurations in the target sector at the stopping level and all of them by $\langle P\rangle = 0.956$. Along its path the number of vortices per configuration only falls, from the 7.3 of the Boltzmann start to below 0.02, as shown in Figure 21. The sector changes while the vortices inherited from the start are being removed and none is built. Direct gradient descent has 15% of its configurations in the target sector at the stopping level and reaches the sector only by building vortices, increasing their number from 7.8 to 8.5 per configuration while the Boltzmann value at the same loss level falls. It still carries 2.1 vortices per configuration at the stopping level where a Boltzmann ensemble has 0.008. Figure 22 shows the resulting change of the topological charge for the individual configurations.

Limitations: the optimized configurations do not form a Boltzmann ensemble in the topological charge. The variance of Q stays at its starting value of about 8 for both methods, whereas a Boltzmann ensemble at the stopping level has 2.7. This is expected rather than a failure of the method. Diffeomorphic optimization moves each sample of the diffusion model individually under a cost function, it does not resample an ensemble, so the optimized collection is not a thermal ensemble at the new loss level. We furthermore note that the sampler crosses sectors freely at this value of β, so that drawing from g until the desired charge is obtained reaches the same sector. The claim

demonstrated here concerns the path taken by the optimization, not the reachability of the target sector.

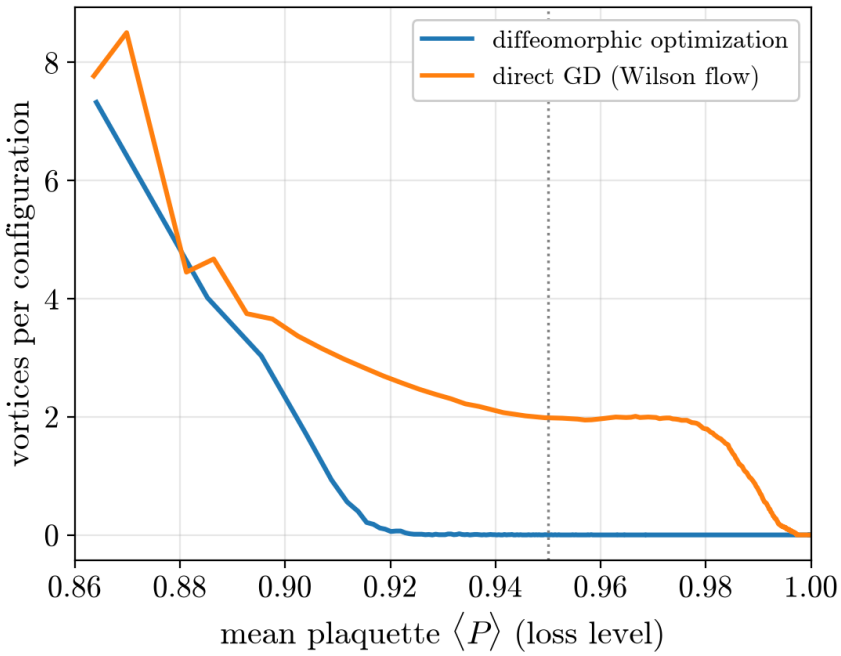


Figure 21: Vortices, i.e. plaquettes with $|\theta_p| > \pi/2$, per configuration along the descent of the two-dimensional $U(1)$ theory. Diffeomorphic optimization changes the topological sector while removing the vortices inherited from the Boltzmann start, whereas direct gradient descent reaches the target sector only by building additional vortices. The two methods start at slightly different values because the inverse of g changes a few configurations.

I Ethics Statement

This work introduces a general optimization framework for generative models and demonstrates its application to protein design. We emphasize that our experiments are limited to computational settings and publicly available protein data. No human or animal subjects are involved, and no wet-lab synthesis or testing is performed.

The methodology has potential implications for drug discovery and bioengineering. While these applications can provide substantial benefits, such as accelerating the design of therapeutic proteins, they also raise concerns about dual use in harmful contexts. We mitigate these risks by focusing exclusively on methodological contributions, using standard benchmark datasets, and refraining from any direct applications to pathogenic proteins or toxins.

Finally, we adhere to principles of reproducibility and transparency. All experiments are conducted with publicly available datasets, and we plan to release code to ensure accessibility and verifiability of our results.

J Reproducibility Statement

We have made efforts to ensure the reproducibility of our results. A detailed description of the mathematical foundation for diffeomorphic optimization algorithm is provided in Section 3, with proofs and theoretical derivations included in Appendix C. The experimental protocols, including optimization settings and evaluation metrics, are described in SectionH.

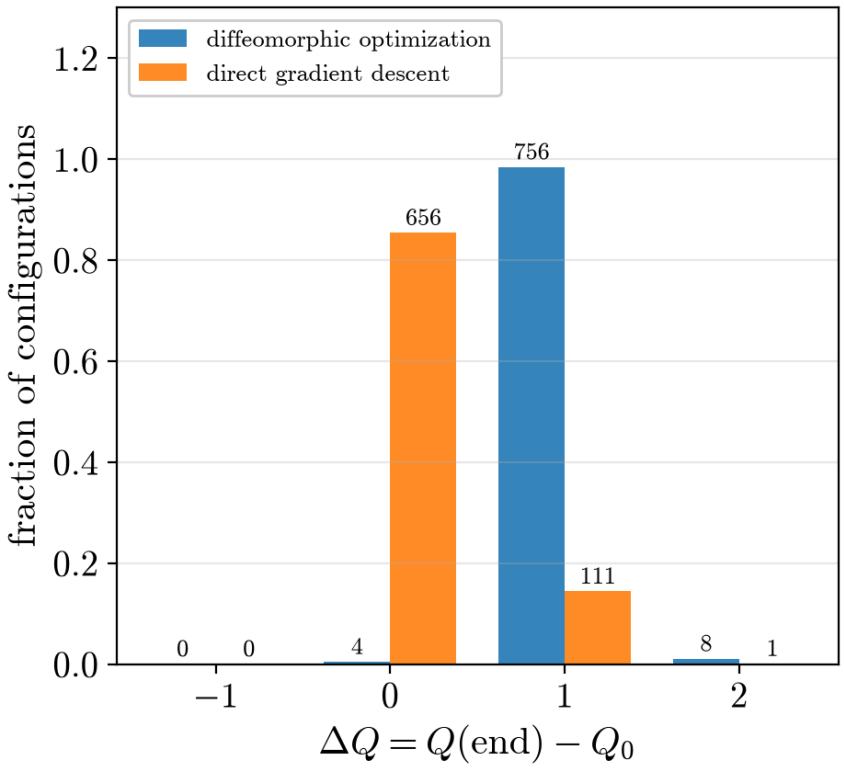


Figure 22: Change of the topological charge Q between the start and the stop of the optimization for the configurations of all three seeds. Diffeomorphic optimization shifts almost all configurations by exactly the requested $+1$, while direct gradient descent moves only a small fraction of them.

All datasets used in this work are publicly available. We describe the data sources and preprocessing steps in Appendix H. For protein experiments, we use standard benchmark datasets from the Protein Data Bank, and we clearly document filtering criteria and feature representations.

The contributions in this work are of theoretical nature. We will release our custom written integration package for Euclidean and matrix Lie groups upon publication, in the hope that it will spur more research.

AI USE STATEMENT

All conceptualization, methodology, code development, experimental analysis, and the creation of figures and results were carried out entirely by the authors. Large language models were not involved in the design of the research, execution of experiments, or interpretation of findings. We used large language models (LLMs) as an aid in polishing the exposition of the manuscript, for example by rephrasing sentences or helping to overcome writer's block. During coding, they were useful for refactoring unwieldy research code.