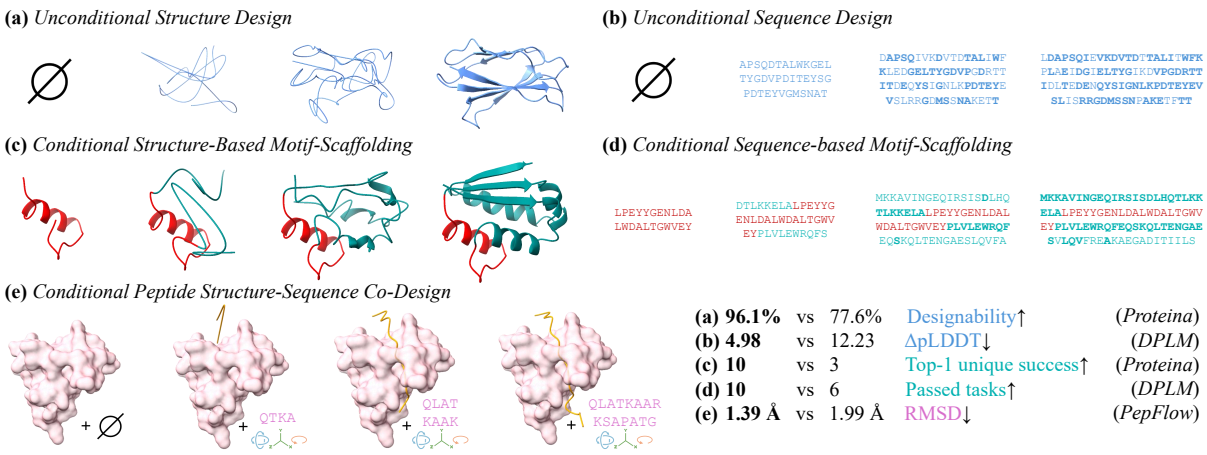
Variable-Length Generative Protein Design via Generalized Poisson Flow

Chaoran Cheng*, **Zhanghan Ni***, **Yanru Qu***, **Yuxin Chen**, **Ruihan Guo**, **Jiajun Fan**, **Ge Liu**
Siebel School of Computing and Data Science
University of Illinois Urbana-Champaign
{chaoran7, zn10, yanruqu2, nealchen, ruihang6, jiajunf3, geliu}@illinois.edu



Abstract

The ability to generate variable-length proteins is crucial in protein design, where the optimal length is often unknown and tightly coupled to designability. Current diffusion- and flow-based generative models typically require the protein length to be specified before sampling, limiting their flexibility in exploring the feasible design space. To address this limitation, we introduce **Generalized Poisson Flow (GPFlow)**, a variable-length generative framework that learns the rate function of an inhomogeneous generalized Poisson process by minimizing its negative log-likelihood. We establish population-level guarantees for recovering the joint multimodal distribution and derive an upper bound on the KL divergence between the data and generated distributions. We comprehensively evaluate GPFlow across structure and sequence design, motif scaffolding, and peptide co-design, spanning Euclidean, categorical, and Riemannian modalities to fully validate its variable-length generation quality. In unconditional design, GPFlow improves structural designability and achieves the best distributional fitness for sequence design compared to fixed-length baselines, while perfectly recovering the length distribution. In conditional motif scaffolding, GPFlow ranks first on 10 of 16 structure-based design tasks with significantly more unique successes and also achieves more passed tasks in sequence-based design. In peptide co-design, GPFlow remains competitive even without access to a native-length oracle.

*Equal contribution.
Correspondence to chaoran7@illinois.edu.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

1 Introduction

Diffusion models [25, 51] and flow matching [39] have become widely used frameworks for protein generative modeling and have produced strong results in protein design [59, 21, 37]. However, their standard formulations operate on fixed-dimensional state spaces. Such a constraint is restrictive for tasks like motif scaffolding, where the optimal length is often unknown *a priori* and is tightly coupled to designability. Sampling sweep over a pre-specified length range increases inference cost and may miss feasible designs when the selected lengths are incompatible with the conditional constraints.

To bridge this gap, we introduce **Generalized Poisson Flow (GPFlow)**, a variable-length generative framework in which length evolves under an inhomogeneous generalized Poisson process. GPFlow couples this length process to a within-length generator, thereby accommodating continuous, discrete, Riemannian, and mixed length-dependent modalities within the same construction. Building on a marginalization theorem for the conditional process, we derive a tractable objective from the negative log-likelihood (NLL) of the stochastic process and further extend the construction to multimodal generation with guarantees on the joint distribution recovery. Furthermore, we derive a generator-based KL upper bound whose terms align with the additive losses, thereby complementing the population-level recovery guarantee at the optimal marginal generator. While the final rate objective aligns with existing work, our formulation provides a common probability-flow construction for continuous, discrete, Riemannian, and mixed modalities, with deeper theoretical connections.

We consider five diverse protein design scenarios: (a) *unconditional structure design* (Section 5.1), (b) *unconditional sequence design* (Section 5.2), (c) *structure-based motif scaffolding* (Section 5.3), (d) *sequence-based motif scaffolding* (Appendix E.4), and (e) *peptide structure-sequence co-design* (Section 5.4). This task landscape evaluates how the same variable-length construction adapts across modalities and generative goals. By retaining the corresponding fixed-length base architectures, we obtain direct comparisons with their variable-length extensions. Across these tasks, our experiments provide evidence that dynamic-length modeling can improve corresponding fixed-length systems in the evaluated settings while removing the need to specify a target length. For unconditional generation, our small structure model significantly improves designability over the corresponding Proteina base model, while the 642M sequence model more closely matches the UniRef50 statistics and maintains higher diversity than the DPLM model. On the motif-scaffolding benchmark, the structure model ranks first on 10 of 16 tasks and yields more than a 10-fold increase in unique successes on several challenging targets, while the sequence model also achieves more passed tasks. In peptide co-design, where translation, rotation, residue types, and torsion angles are learned jointly with insertion, GPFlow remains competitive and outperforms the PepFlow base model even without access to a native-length oracle. Our contributions are summarized as follows:

- **A unified variable-dimensional probability flow.** We formulate the variable-length generative modeling in GPFlow with inhomogeneous generalized Poisson probability paths and couple them with continuous, discrete, Riemannian, or mixed within-length dynamics. A single marginalization principle covers these modality types for multimodal generation.
- **Simulation-free likelihood learning with theoretical guarantees.** We derive the rate objective from the generalized Poisson trajectory likelihood while analytically marginalizing its auxiliary insertion times, so training is fully simulation-free. We further prove population-level recovery of the joint multimodal distribution and establish a generator-based KL upper bound.
- **Variable-length generation across five protein-design settings.** We instantiate the framework for protein structure design, sequence design, motif scaffolding, and peptide structure-sequence co-design. Across all tasks, direct comparisons with the fixed-length base model demonstrate GPFlow's superior performance and flexibility. We achieve superior structural designability, better sequence fitness, more unique successes and passed tasks in motif scaffolding, and stable peptide designs without a native-length oracle.

2 Related Work

Protein Design Models Building on diffusion and flow matching, recent generative models for protein design have achieved strong performance. FrameDiff [60] adopts Riemannian diffusion to learn the positions and orientations of each amino acid, and FrameFlow [59] further extends it with Riemannian flow matching. Proteina [21] is a CA-only flow-based structure generative model that achieves remarkable scaling results. RFDiffusion [58, 9] is a family of diffusion-based models

that incorporates the structure-prediction capabilities of RoseTTAFold [3] into generative modeling. Generally, these structure generative models also support motif scaffolding, in which fragments of functionally important protein structures, known as *motifs*, are provided to the model as conditions. There are also dedicated models for motif scaffolding, e.g., Genie2 [37] is a diffusion-based generative model with a specialized multi-motif framework that designs co-occurring motifs. These models either adopt an inpainting-style sampling that enforces hard constraints by fixing the motif residues and structures, or inject motif information as an explicit but soft conditioning signal. Additional design models for sequences and peptides are available in Appendix B.6.

Variable-Length Generative Models Standard diffusion or flow matching models require a predetermined sampling length, limiting their applicability to variable-length data. Two existing works incorporate data length through related stochastic-process constructions. TDDM [10] models dimension changes with a learnable jump kernel coupled to continuous diffusion dynamics, but does not treat a categorical length-dependent modality. EditFlow [23] instead derives insertion rates from discrete flow matching [19]; it applies to categorical modalities but not to continuous coordinates. Importantly, neither of these models has been applied to the protein design domain. More recently, SCISOR [4] approaches variable-length protein sequence generation via a deletion-based process, where random residues are inserted during forward noising and removed by the learned reverse process. Concurrent Branching Flow [6] constructs variable-length probability paths via deterministic branching of mean positions and applies it to latent protein generation and small-molecule co-design. Additional related work can be found in Appendix B.6.

GPFlow differs primarily in scope and probabilistic interpretation. It uses one unified variable-dimensional probability-flow formulation for continuous, discrete, Riemannian, and mixed modalities. Our rate objective follows from the generalized Poisson trajectory likelihood, in which auxiliary insertion times are analytically marginalized to yield a simulation-free objective. In their overlapping regimes, GPFlow recovers the insertion-only, modality-specific instantiations of TDDM and EditFlow and yields the same rate objective. Our contribution is therefore the unified construction and likelihood interpretation with the new KL bound on the training dynamics. While the original TDDM and EditFlow papers do not evaluate protein design tasks, our experiments instantiate the framework across five comprehensive settings for unconditional and conditional protein design.

3 Generalized Poisson Flow

To construct a probability path $\{p_t\}_{t\in[0,1]}$ for a stochastic state Y_t using flow matching (or diffusion), one needs (a) $p_1 \approx q$, where q is the data distribution, (b) simulation-free sampling from p_t for feasible training, and (c) a tractable loss for learning the marginal flow. We first derive theoretical results for the length-only state $Y_t = X_t \in \mathbb{N}$ and then extend them to the multimodal state $Y_t = (X_t, S_t) \in \mathsf{Y}$, where S_t may comprise arbitrary discrete or continuous length-dependent modalities or their mixes. We demonstrate how to construct feasible probability paths using the generalized Poisson process.

3.1 Length-Only Generalized Poisson Flow

A *generalized Poisson process* $\{X_t\}_{0\le t\le T}$ is a continuous-time Markov process defined by a *rate function* $\lambda_t : [0,T]\times\mathbb{N}\to\mathbb{R}^{\ge 0}$, such that $\Pr(X_{t+h} = X_t + 1) = \lambda_t(X_t)h + o(h)$ and $\Pr(X_{t+h} = X_t) = 1 - \lambda_t(X_t)h + o(h)$. Consistent with the flow matching convention, we will assume the generalized Poisson process to be defined on the time interval $[0,1]$ with $X_0 = 0$. We now show one feasible construction of the probability path $p_t(X)$ using the generalized Poisson process with the following theorem:

Theorem 3.1. *Let $\kappa : [0,1]\to[0,1]$ be a nondecreasing, absolutely continuous scheduler with $\kappa_0 = 0$ and $\kappa_1 = 1$, and define the* completion time *$\tau_\kappa := \inf\{t : \kappa_t = 1\}$. For any fixed $L \ge 1$, assume $\kappa_t < 1$ for $t < \tau_\kappa$ and define the insertion* hazard *and* conditional rate *by*

$$h_t := \begin{cases} \dfrac{\dot\kappa_t}{1-\kappa_t}, & t < \tau_\kappa, \\ 0, & t \ge \tau_\kappa, \end{cases} \qquad \lambda_t(X_t|L) := (L - X_t)h_t. \tag{1}$$

Then, the resulting process satisfies $X_t \sim B(L,\kappa_t)$ for $t < \tau_\kappa$ and is absorbed at $X_t = L$ for $t \ge \tau_\kappa$.

See the proof in Appendix A.1. Theorem 3.1 offers a simulation-free approach for sampling X_t and guarantees $X_1 = L$ almost surely. Similar to standard flow matching for continuous modalities, we then establish the following marginalization theorem:

Theorem 3.2. *For any scheduler κ_t defined above, let q be a target length distribution, define $p_t(k|y) := \Pr(X_t = k|X_1 = y)$, and let $p_t(k) := \sum_y p_t(k|y)q(y)$. For every k with $p_t(k) > 0$, the posterior probability is $p(X_1 = y|X_t = k) := p_t(k|y)q(y)/p_t(k)$. Then the* marginal rate

$$\lambda_t^*(k) := \mathbb{E}_{X_1 \sim p(\cdot|X_t=k)}[\lambda_t(k|X_1)] \tag{2}$$

generates the marginal path p_t and hence $p_1 = q$.

See Appendix A.2 for the proof, which relies primarily on the Kolmogorov forward equation as the continuity equation. Theorem 3.2 guarantees that any length distribution q can be modeled using a generalized Poisson process and also ensures the data constraint $p_1 = q$ for constructing a probability flow p_t over the discrete length space. We therefore refer to our generative framework as *Generalized Poisson Flow*. To effectively learn the flow, in Proposition A.1, we prove that the negative log-likelihood of a realization of a generalized Poisson process can be calculated as $\mathcal{L}_{\text{NLL}} = \int_0^1 \lambda_t \, \mathrm{d}t - \sum_{k=1}^L \log \lambda_{t_k}$, where the two terms can be viewed as the NLL for the survival and event terms, respectively. The insertion times are auxiliary variables used to construct the probability path, and by the integration formula for a stochastic point process [8], the expected sum over insertion times can be marginalized into an integral against the ground-truth marginal rate λ_t^*. Combining the marginalization result in Theorem 3.2 and the conditional rate in Eq. 1, we have:

$$\mathbb{E}_{\{t_k\}_{k=1}^L, L}\left[-\sum_{k=1}^{L} \log \lambda_{t_k}\right] = \mathbb{E}_t[-\lambda_t^* \log \lambda_t] = \mathbb{E}_{t, X_1 \sim q(X)}\left[-h_t(X_1 - X_t) \log \lambda_t\right]. \tag{3}$$

Therefore, we arrive at a tractable, simulation-free NLL loss that requires only an endpoint X_1, a sampled time t, and the corresponding intermediate state X_t obtained from Theorem 3.1:

$$\mathcal{L}_{\text{GP}} = \mathbb{E}_{t, X_1 \sim q(X)}\left[\lambda_t - h_t(X_1 - X_t) \log \lambda_t\right]. \tag{4}$$

We note that, unlike continuous flow matching, in which the ground-truth marginal vector field is intractable, Theorem 3.1 provides a closed-form conditional rate target.

3.2 Multimodal Generalized Poisson Flow

To truly unlock variable-length protein structure and sequence design, we now extend the length-only formulation to incorporate other length-dependent modalities such as continuous coordinates or discrete types. Let S be a component space and define the variable-length state space $\mathsf{Y} := \bigsqcup_{k \geq 0} \{k\} \times \mathsf{S}^k$. The component space may be continuous (with a Lebesgue or manifold-volume reference measure), discrete (with counting measure), or a product of such spaces. We write $Y_t = (X_t, S_t) \in \mathsf{Y}$, where $S_t = (S_t^1, \ldots, S_t^{X_t})$. An *insertion* of a component can be described by a rate λ_t and a sampling distribution ρ_t that selects the new component value (and, in the ordered case, its insertion position). Between insertions, a within-length generator $\mathcal{B}_t$ refines the existing components. For continuous modalities, $\mathcal{B}_t f = v_t \cdot \nabla f$ (with an optional diffusion term); for discrete modalities, $\mathcal{B}_t$ is the corresponding continuous-time Markov-chain generator.

The key observation is that the conditional generators associated with individual clean targets can be marginalized under the posterior of the target given the current state. The following theorem states this construction; a measure-theoretic formulation and proof are provided in Appendix A.4.

Theorem 3.3. *Let $Y_1 = (X_1, S_1) \sim q$ index conditional probability paths $p_t(\cdot|Y_1)$ on Y, with $p_1(\cdot|Y_1) = \delta_{Y_1}$. At state $y \in \mathsf{Y}$, suppose each conditional process has within-length generator $\mathcal{B}_t^{Y_1}$, insertion rate $\lambda_t(y|Y_1)$, and sampling distribution $\rho_t(\mathrm{d}m|y, Y_1)$ for the insertion variable m. For any y with $p_t(y) > 0$, the posterior of the clean target is $p(\mathrm{d}y_1|Y_t = y) := p_t(y|y_1)q(\mathrm{d}y_1)/p_t(y)$, where $p_t = \int p_t(\cdot|y_1)q(\mathrm{d}y_1)$. Define the marginals by expectations over this posterior:*

$$\mathcal{B}_t^* f(y) := \mathbb{E}_{Y_1 \sim p(\cdot|Y_t=y)}[\mathcal{B}_t^{Y_1} f(y)], \quad \lambda_t^*(y) := \mathbb{E}_{Y_1 \sim p(\cdot|Y_t=y)}[\lambda_t(y|Y_1)], \tag{5}$$

$$\rho_t^*(\mathrm{d}m|y) := \frac{\mathbb{E}_{Y_1 \sim p(\cdot|Y_t=y)}[\lambda_t(y|Y_1)\rho_t(\mathrm{d}m|y, Y_1)]}{\mathbb{E}_{Y_1 \sim p(\cdot|Y_t=y)}[\lambda_t(y|Y_1)]}, \tag{6}$$

where ρ_t^* can be arbitrary when the marginal rate is zero. Then the process with $\mathcal{B}_t^*$ and insertion kernel $Q_t^* = \lambda_t^* \rho_t^*$ generates the marginal p_t and hence $p_1 = q$.

The proof follows the same posterior-marginalization principle as Theorem 3.2. In the continuous case, $\mathcal{B}_t^*$ reduces to the usual marginal vector field; in the discrete case, the same posterior expectation marginalizes the within-length transition rates. We learn λ_t^* using the Poisson NLL in Eq. 4. The within-length generator is learned with the appropriate continuous or discrete flow-matching loss $\mathcal{L}_{\text{FM}}$. As the sampling distribution $\rho_t(\cdot|y, Y_1)$ is a mixture over the $X_1 - X_t$ upcoming insertions, the general reconstruction NLL is:

$$\mathcal{L}_{\text{rec}} = \mathbb{E}_{Y_1, Y_t}\left[-h_t \sum_{k=1}^{X_1 - X_t} \log \rho_t(Y_t, s_k)\right], \tag{7}$$

where the marginal rate term $(X_1 - X_t)h_t$ is combined with the marginal length denominator of $1/(X_1 - X_t)$ to give the final coefficient. Generally, the final loss for multimodal GPFlow is

$$\mathcal{L} = \mathcal{L}_{\text{GP}} + w_{\text{rec}}\mathcal{L}_{\text{rec}} + w_{\text{FM}}\mathcal{L}_{\text{FM}}, \tag{8}$$

where $w_{\text{rec}}, w_{\text{FM}}$ are hyperparameters. We describe the parameterization choices for learning the flow and sampling distributions in Appendix B.3 and B.4. Sampling from a multimodal GPFlow generally follows the procedure discussed above, in which flow sampling for the existing length-dependent modalities is performed using the standard approach with the learned $\mathcal{B}_t$. An event is then sampled according to the learned rate λ_t, and if successful, it will draw the new insertion value from the learned sampling distribution $s_{\text{new}} \sim \rho_t$. We outline the sampling procedure in Algorithm 1.

In practice, because proteins have a natural ordering according to the residue index, we need to extend our formulation to an order-preserving multimodal GPFlow. Intuitively, this can be easily obtained by modeling $X_t + 1$ separate rates and sampling distributions to represent the different insertion positions while preserving the order. A more rigorous formulation together with a modified characterization of the sampling distribution is available in Appendix B.1.

Algorithm 1 Sampling from Multimodal GPFlow (Euler)

$Y_0 = (X_0 = 0, S_0 = \emptyset)$.
for $i \leftarrow 1, 2, \ldots, N$ **do**
  $t \leftarrow t_i, \Delta t \leftarrow t_{i+1} - t_i$, where $t_{N+1} := 1$.
  Predict the rate $\lambda_\theta(Y_t, t)$, the vector field $v_\theta(Y_t, t)$, and the sampling distribution $\rho_\theta(Y_t, t)$.
  Advance the generative flow for each S_t with v_θ to obtain $\hat{S}_{t+\Delta t}$.
  Simulate the Poisson process by sampling an event on $X_{t+\Delta t}$ with probability $\lambda_\theta \Delta t$.
  If an event happens, insert the new feature value $s_{\text{new}} \sim \rho_\theta$ into $\hat{S}_{t+\Delta t}$ to obtain $S_{t+\Delta t}$.
  Update $Y_{t+\Delta t} = (X_{t+\Delta t}, S_{t+\Delta t})$.
Return: $Y_1 = (X_1, S_1)$.

Adapting to conditional generation is generally straightforward for GPFlow and does not affect the theoretical results. In the multimodal setup, the condition parts $Y_0 = (X_c, S_c)$ are sampled from the clean conditioning data and remain unchanged along the conditional probability path. During sampling, instead of starting from an empty list, we initialize from the provided conditions and their length-dependent modality, mimicking the standard "inpainting" setup. Depending on the task, we may or may not allow insertions between the conditions. For example, in the motif scaffolding setup, insertion is not allowed inside a contiguous motif part, but is allowed between different motifs and at the beginning and end.

4 Theoretical Guarantees on Training Dynamics

In this section, we derive a generator-based upper bound on the KL divergence between the ground-truth and generated distributions and relate its terms to the exact likelihood components of our loss objective. We consider general Markov processes as a superset of GPFlow. We first introduce the *generator* as a useful tool for describing the time evolution of their probability densities, and then derive a bound on the KL divergence between the distributions generated by two Markov processes. According to Holderrieth et al. [26], under some regularity assumptions, a Markov process $\{X_t\}_{t\geq 0}$ can be characterized with its generator $\mathcal{A}_t f(X_t) = \lim_{h\to 0^+}(\mathbb{E}[f(X_{t+h})|X_t] - f(X_t))/h$, which can be determined by the velocity field u_t, the diffusion coefficient Σ_t, and the jump measure Q_t as:

$$\mathcal{A}_t f(x) = u_t(x) \cdot \nabla f(x) + \frac{1}{2}\text{Tr}\left(\Sigma_t(x)\nabla^2 f(x)\right) + \int [f(y) - f(x)]\, Q_t(\mathrm{d}y|x). \tag{9}$$

If all X_t's have a probability density function p_t, then $\mathbb{E}_{x\sim p_t}[\mathcal{A}_t f(x)] = \int \dot{p}_t(x) f(x)\,\mathrm{d}x$ connects the generator with the time evolution of the density p_t. In common diffusion models, we consider a Markov process where $\Sigma_t = \sigma_t^2 I$ for some scalar function σ_t.

4.1 KL Divergence Bound for Markov Processes

We now derive a generator-based bound on the KL divergence between the distributions generated by two Markov processes, which generalizes the variable-length result in Campbell et al. [10]. See Appendix A.6 for a detailed proof.

Theorem 4.1. *Let $\{X_t\}_{t\geq 0}$ and $\{\tilde{X}_t\}_{t\geq 0}$ be two Markov processes with generators $\mathcal{A}_t$ and $\tilde{\mathcal{A}}_t$, respectively. Suppose $\mathcal{A}_t$ is determined by (u_t, σ_t^2, Q_t) and $\tilde{\mathcal{A}}_t$ is determined by $(\tilde{u}_t, \sigma_t^2, \tilde{Q}_t)$, so they coincide in the diffusion coefficient. Assume that both processes have the same initial distribution p_0 and that sufficient regularity conditions hold. Then, the KL divergence between p_T and $\tilde{p}_T$ can be bounded by the loss terms $\mathcal{L}_t(x)$ as*

$$\mathcal{D}_{KL}(p_T\|\tilde{p}_T) \leq \int_0^T \mathbb{E}_{x\sim p_t}\left[\mathcal{L}_t(x)\right]\mathrm{d}t, \quad \mathcal{L}_t(x) = \frac{\|u_t(x) - \tilde{u}_t(x)\|_2^2}{2\sigma_t^2} + \mathcal{D}_{KL}(Q_t\|\tilde{Q}_t), \tag{10}$$

where the first drift-diffusion term is omitted when no continuous component is present, and $\mathcal{D}_{KL}(Q_t\|\tilde{Q}_t)$ denotes the generalized KL divergence between two positive measures:

$$\mathcal{D}_{KL}(Q\|\tilde{Q}) = \int \tilde{Q}(\mathrm{d}y|x) + \int \left(\log \frac{\mathrm{d}Q(y|x)}{\mathrm{d}\tilde{Q}(y|x)} - 1\right) Q(\mathrm{d}y|x). \tag{11}$$

4.2 Generator View of GPFlow

We now discuss how to interpret GPFlow as a special case of a continuous-time generator on the joint state space Y. The marginal within-length generator $\mathcal{B}_t^*$ that refines existing components constitutes the first drift-diffusion term in the continuous case or the generalized KL term of the transition kernel for the finite Markov chain in the discrete case [19]. On the other hand, the marginal jump measure across dimensions factorizes as $Q_t^* = \lambda_t^* \rho_t^*$: at state y, an event occurs at rate $\lambda_t^*(y)$, after which the new component (and insertion position, when applicable) is sampled from $\rho_t^*(\mathrm{d}m|y)$. The learned counterpart is the jump kernel $Q_\theta = \lambda_\theta \rho_\theta$, factorized into rate $\lambda_\theta(y,t)$ and sampling distribution $\rho_\theta(\mathrm{d}m|y,t)$. See Appendix A.7 for the precise state-space construction and the explicit generator definition. One crucial observation is that the gradient of our NLL-based rate loss objective coincides with the gradient of the expected generalized KL divergence:

$$\nabla_\theta \mathcal{L}_{\text{GP}} = \nabla_\theta \mathbb{E}_{t,Y_t\sim p_t}\left[\mathcal{D}_{\text{KL}}\big(\lambda_t^*(Y_t)\|\lambda_\theta(Y_t,t)\big)\right]. \tag{12}$$

In Appendix A.7, we describe how the jump contribution decomposes exactly into a generalized KL for the rate and a rate-weighted KL for the sampling distribution. Therefore, Theorem 4.1 is applicable and implies that the additive GPFlow objective in Eq. 8 minimizes the corresponding upper-bound terms, thereby providing a finer-grained distributional guarantee on the training dynamics.

Additionally, our theoretical framework offers a natural explanation of why the generalized KL divergence is chosen over other Bregman divergences, such as MSE. Importantly, existing works' choice of the generalized KL divergence as the rate loss is purely empirical. In contrast, our choice is derived directly from the exact point process NLL, thereby connecting variable-length generative loss to the underlying, more principled stochastic process NLL minimization.

5 Experimental Results

We instantiate our GPFlow across five diverse protein design tasks across continuous, discrete, and Riemannian modalities. Additional details on datasets, models, and pipelines can be found in Appendix C. Additional ablation studies and experimental results are in Appendix D and E.

5.1 Unconditional Protein Structure Generation

We first instantiate GPFlow for unconditional structure generation, in which the CA coordinates serve as the continuous length-dependent modality. We follow the setup in Proteina [21], a CA-only

protein generative model. We build our order-preserving GPFlow on the 60M Proteina model, adding additional heads to predict rates and reconstruct sampling distributions for the continuous coordinates, bringing the total to 65M trainable parameters. The datasets also follow the Proteina setup, including PDB [5] and AFDB [29]. For PDB, we retain only single-chain entries with lengths between 50 and 256, yielding 27k data samples in total. For AFDB, we follow Proteina's clustering instructions to obtain 713k data samples. We train two GPFlow variants on the two datasets from scratch. For the Proteina baseline, we use its official 60M checkpoint trained on AFDB for inference and retrain a model on PDB using the same hyperparameters.

For the fixed-length models, we follow the standard Proteina benchmark to sample 100 structures at each length in $\{50, 100, 150, 200, 250\}$. For GPFlow, we generate 512 structures without a length input; their lengths are generated from the learned rate. We report **Designability** as the proportion of *designable* structures averaged across all generated samples. A designable structure is defined as having a best self-consistent RMSD (scRMSD) less than 2 Å to the 8 ProteinMPNN [16] redesigned and ESMFold [38] refolded candidates. Additional diversity-based metrics and secondary-structure ratios are also reported. **Diversity** is defined as the proportion of the designable clusters in all designable samples. **Novelty** is defined as the maximum TM-score [61] via exhaustive search against the PDB or AFDB databases, averaged across all samples. Therefore, a higher diversity score indicates more designable clusters, and a lower novelty score indicates more novel generations.

Table 1: Unconditional protein structure design benchmark. The best result for each metric is in **bold**.

Model	Design-ability ↑	Diversity ↑	Novelty ↓		Struct% α/β
			PDB	AFDB	
FrameDiff [60]	65.4	0.39	0.73	0.75	64.9/11.2
FrameFlow [59]	88.6	0.53	**0.69**	**0.73**	55.7/18.4
RFDiffusion [58]	94.4	0.46	0.71	0.77	64.3/17.2
Proteina* (PDB)	77.6	0.25	0.78	0.81	48.7/15.2
Proteina† (AFDB)	90.6	**0.57**	0.77	0.84	68.8/6.3
GPFlow (PDB)	**96.1**	0.26	0.76	0.80	61.2/13.8
GPFlow (AFDB)	94.5	0.43	0.75	0.82	72.4/5.4

*A 60M Proteina is retrained on PDB.
†Official Proteina 60M checkpoint for inference.

The results are summarized in Table 1, where GPFlow achieves a 3.9%–18.5% improvement in designability over the corresponding Proteina base model. Figure 1 serves as an intrinsic calibration check for the learned length marginal: GPFlow's generated distribution follows the empirical training profile with a mean length of 153.7, whereas the fixed-length benchmark is 150. Appendix E further reports designability across GPFlow's generated length bins, providing more concrete evidence that our superior designability is not due to the unmatched length distribution but to its better capture of the joint distribution. In terms of novelty, we note that GPFlow achieves a lower TM-score than the corresponding Proteina model, indicating greater novelty, although both models fall behind other baselines. The generally lower diversity of the Proteina-family models, including GPFlow, is due to the inductive bias of low-temperature sampling, further ablated in Appendix D.2. The motif scaffolding benchmark provides a more concrete evaluation of the diversity of GPFlow through the number of unique successes after clustering.

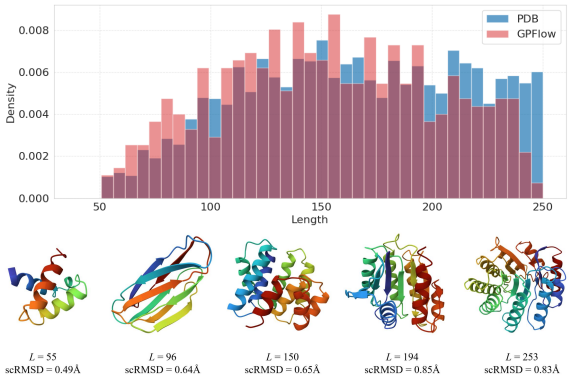


Figure 1: Length distributions and representative examples of GPFlow-generated structures.

5.2 Unconditional Protein Sequence Generation

We instantiate GPFlow to generate variable-length protein sequences, incorporating the amino acid sequence as the discrete length-dependent modality. We train GPFlow on 41M UniRef50 [53] sequences of lengths 10-1024 with a 642M-parameter DiT [44]. To improve long-sequence generation, we follow Havasi et al. [23] to adopt a localized probability path as detailed in Appendix C.2.5. For inference-time correction, we additionally learn a deletion process that can be coupled with the insertion process during sampling. Three different paradigms of protein sequence generative models are compared. For fixed-length discrete diffusion over categorical amino-acid tokens, we evaluate EvoDiff-OADM [1] and DPLM [57]. For masked language models with iterative decoding,

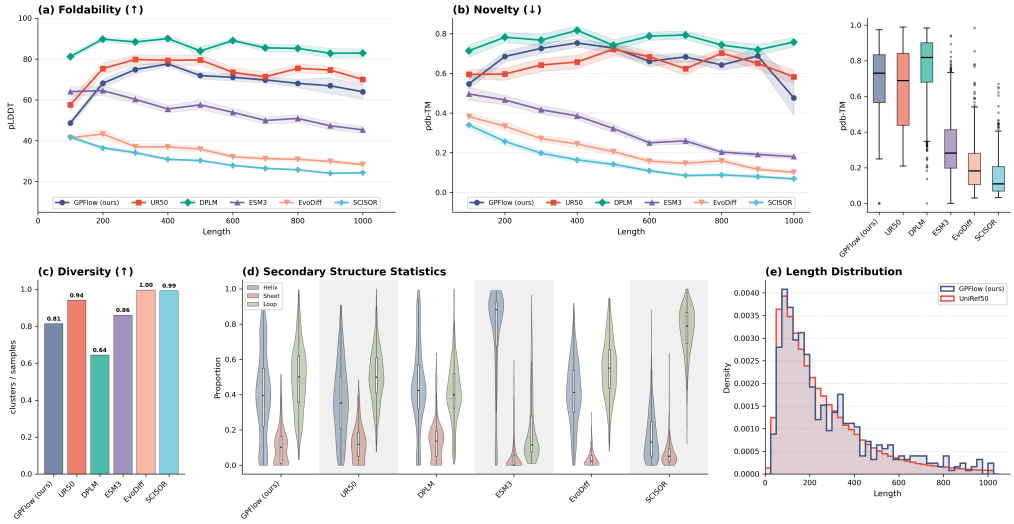
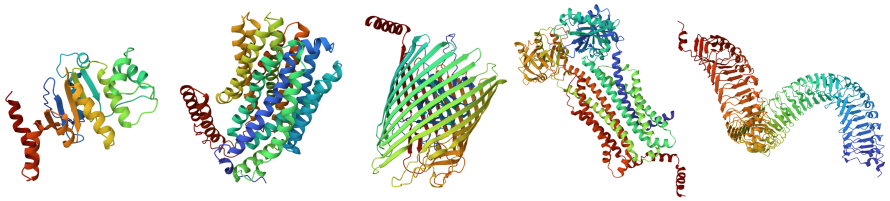


Figure 2: Performance benchmark for unconditional protein sequence design. **(a)** Foldability (pLDDT) across sequence length bins. **(b)** Novelty measured by TM-score to the nearest PDB structure. **(c)** Diversity by FoldSeek clustering. **(d)** Secondary structure distributions. **(e)** Length distribution of GPFlow samples generated without length conditioning. Error bands denote ± 1 SEM.

we evaluate ESM3 [24] restricted to its sequence track. For variable-length generation, we evaluate SCISOR [4], a deletion-based discrete model, as a special case of EditFlow [23]. While other baselines are trained on UniRef50, ESM3 is trained on a substantially larger corpus that subsumes UniRef50 and is included for reference. To evaluate the generated samples, we fold the generated sequences with ESMFold [38] and quantify: (a) **Foldability** via mean pLDDT; (b) **Novelty** via `pdb-TM`, the maximum TM-score to any PDB structure found by FoldSeek search (TM < 0.5); (c) **Diversity** via FoldSeek clustering at TM ≥ 0.5, reported as the proportion of unique clusters among all generations; (d) **Secondary structure** composition via DSSP (helix/strand/coil fractions) [30]; and (e) **Length distribution**, compared to UniRef50.

Figure 2(a-d) evaluates structural statistics when assessed by ESMFold. Across all lengths, GPFlow is the closest match to UniRef50 in mean pLDDT, faithfully reproducing the structural characteristics rather than optimizing toward uniformly high-confidence folds. DPLM achieves an abnormally high pLDDT compared to the training data, with a mean pLDDT difference of 12.23, whereas GPFlow achieves 4.98. DPLM has the lowest structural novelty and diversity among the models, illustrating a clear trade-off and off-data behavior, whereas GPFlow maintains 17% higher diversity while keeping novelty closest to UniRef50. In contrast, ESM3, EvoDiff, and SCISOR exhibit consistently lower pLDDT together with lower `pdb-TM`. Finally, GPFlow best matches UniRef50 in secondary-structure composition, including a near-identical mean structural balance. DPLM deviates from the UniRef50 profile in loop content, whereas ESM3, EvoDiff, and SCISOR exhibit large departures from the UniRef50 profile.

Figure 2(e) illustrates the variable-length formulation: GPFlow learns and samples from an emergent length prior that closely matches the training length. Fixed-length baselines define only $p_\theta(x|L)$ and therefore have no marginal length distribution without an additional sampler over L. Selected generated sequences folded by ESMFold are presented in Figure 3 with their lengths and pLDDTs.



$L = 217$ pLDDT = 82.1 | $L = 482$ pLDDT = 81.8 | $L = 686$ pLDDT = 76.4 | $L = 865$ pLDDT = 80.4 | $L = 954$ pLDDT = 89.9

Figure 3: Representative unconditional sequence generations folded by ESMFold.

5.3 Motif Scaffolding

Table 2: Number of unique successes (1000 generations for each task) on the RFDiffusion benchmark [58]. Superscripts denote the number of motif segments. Motif tasks with multiple lengths are indicated in parentheses. Best results are shown in **bold**.

Task Name (contig ver.)	GPFlow (Ours)	Proteina [21]	Genie2 [37]	RFDiffusion [58]	FrameFlow [59]
Unique success (avg. / total)	**117.4 / 1878**	94.9 / 1519	64.8 / 1037	41.5 / 664	51.1 / 817
1BCF[4]	**11**	1	1	1	1
1PRW[2]	**42**	1	1	1	1
1QJG[3]	1	3	5	1	**18**
1YCR[1]	**413**	249	134	7	149
2KL8[2]	**3**	1	1	1	1
3IXT[1]	**398**	8	14	3	8
4JHW[2]	0	0	0	0	0
4ZYP[1]	**144**	11	3	6	4
5IUS[2]	**3**	1	1	1	0
5TPN[1]	**147**	4	8	5	6
5TRV[1] (long / med / short)	136	**179** / 22 / 1	97 / 23 / 3	23 / 10 / 1	77 / 21 / 1
5WN9[1]	**19**	2	1	0	3
5YUI[3]	**9**	5	3	1	1
6E6R[1] (long / med / short)	268	**713** / 417 / 56	415 / 272 / 26	381 / 151 / 23	110 / 99 / 25
6EXZ[1] (long / med / short)	263	290 / 43 / 3	326 / 54 / 2	167 / 25 / 1	**403** / 110 / 3
7MRX[1] (128 / 85 / 60)	21	51 / 31 / 2	27 / 23 / 5	**66** / 13 / 1	35 / 22 / 1

For conditional generation, we instantiate GPFlow for motif scaffolding, with the motif structure and/or sequence as the conditioning variable. Starting from isolated motif segments, our model gradually grows a protein scaffold through inserting and updating linker residues between motif segments, without relying on predefined contig templates. We consider both structure-based and sequence-based motif scaffolding tasks. The latter is deferred to Appendix E.4.

For structure-based motif scaffolding, we build our GPFlow upon the Proteina variant, which injects motif structure and residue information as conditioning signals and fixes the motif portion during both training and sampling. We train it from scratch on the same AFDB dataset and adopt the motif training augmentation used in Proteina. The evaluation criteria follow Watson et al. [58], which uses a combination of scRMSD, motifRMSD, pLDDT, and pAE thresholds detailed in Appendix C.1.3. All successes are clustered via FoldSeek to count unique successes rather than raw successes.

Table 2 lists the unique success counts across 16 motif tasks from the RFDiffusion benchmark [58], where different contig templates corresponding to the same motif target are grouped into a single task for GPFlow. Overall, GPFlow ranks first on 10 out of 16 motif tasks. For easier tasks like 1BCF and 3IXT, GPFlow yields considerably more diverse generations, with variation in both secondary structures and scaffold lengths, whereas the base Proteina generations collapse into only a few clusters. (Figure 4). For more challenging tasks like 5WN9 and 5YUI, the raw success counts of GPFlow also dominate. Noticeably, GPFlow can also generate successful scaffolds longer than 200 residues, whereas Proteina generations are restricted by the evaluated contig templates. A finer-grained analysis of the generation lengths in Appendix E.3 further confirms the diversity of our generations across different length bins. Our evaluations show that the learned length-varying process can produce successful scaffolds over a wider range than the predefined contig choices used by the fixed-length baselines.

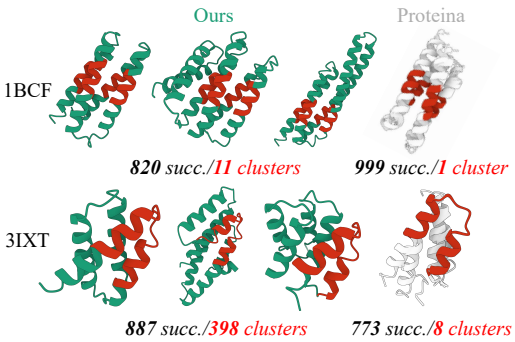


Figure 4: GPFlow generates more diverse clusters for the same motif (highlighted in red).

5.4 Peptide Co-Design

Finally, we extend GPFlow to peptide sequence-structure co-design. Given a target receptor protein, the model jointly generates the all-atom structure and amino-acid sequence of a binding peptide,

with the peptide length not predetermined. We use the dataset and benchmark from Li et al. [34], in which the receptor structure and pocket positions are provided as conditioning context. GPFlow is adapted from the PepFlow [34] architecture, in which per-residue rotation, translation, residue type, and side-chain torsion angles are learned jointly to model the all-atom peptide structure. Rotation and torsion angles are modeled using Riemannian flow matching, with an additional extension that learns inserted elements using the corresponding Riemannian norm. The residue types are modeled using a simplex-based continuous representation that embeds soft one-hot logits in Euclidean geometry.

We compare against the three co-design baselines: RFdiffusion [58] composed with ProteinMPNN [16] for sequence design, ProteinGenerator [40] that jointly samples backbone and sequence, and the base PepFlow model. We report eight generative metrics: amino-acid recovery (**AAR**), **RMSD**, secondary-structure ratio (**SSR**), binding-site recovery (**BSR**), **Affinity**, **Stability**, **Designability** and **Diversity**. The complete evaluation protocol and the length-matched/unmatched breakdown are provided in Appendix C.3.3. As shown in Table 3, GPFlow attains the best performance on 4 out of the 8 metrics and also surpasses the base PepFlow in terms of Designability and Diversity. Notably, GPFlow is the only model that samples peptide length from a learned rate function rather than conditioning on the native length (Appendix E.5). This matters for *de novo* peptide design, where fixed-length baselines rely on a native-length oracle unavailable in practice. Figure 5 highlights two qualitative behaviors of GPFlow with a better energy landscape. In 1cwd_P, the processed ground truth contains a gap induced by the removal of a non-standard residue, and PepFlow inherits this gap through its native-indexed generation. In contrast, GPFlow generates a continuous peptide across this region. In 2qlj_E, the generated insertion extends more deeply into the binding site, suggesting greater sensitivity to the local pocket geometry.

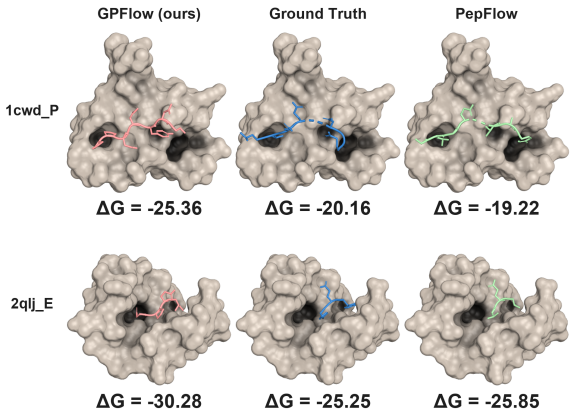


Figure 5: Samples of GPFlow-generated peptides compared to ground truth and PepFlow [34].

Table 3: Conditional sequence-structure co-design on peptide design. Best results are shown in **bold**. † are composite over the GT-length matched/unmatched samples; see Appendix C.3.3 for details.

Model	Geometry				Energy		Design		Length	
	AAR % ↑	RMSD Å ↓	SSR % ↑	BSR % ↑	Affinity % ↑	Stability % ↑	Designability % ↑	Diversity ↑	Mean	Requires GT?
ProteinGenerator [40]	28.18	4.36	58.89	15.60	11.25	**40.00**	72.66	**0.476**	10.53	✓
RFdiffusion+MPNN [58]	26.69	4.17	62.70	18.09	7.27	17.36	**84.72**	0.408	10.53	✓
PepFlow [34]	46.35	1.99	78.98	**89.40**	11.74	6.46	37.27	0.390	10.53	✓
GPFlow	**53.71**†	**1.39**†	**91.08**†	86.04	**21.03**	6.27	48.19	0.443	9.86	✗

6 Conclusion & Limitations

In this work, we propose GPFlow, a unified framework for variable-length generative modeling applicable to diverse protein design scenarios. GPFlow uses a generalized Poisson process coupled to within-length dynamics and covers multimodal settings with continuous, discrete, and mixed modalities. Simulation-free rate learning based on exact negative log-likelihood minimization is proven via posterior marginalization, with an additional generator-based KL bound on the training dynamics. Our comprehensive experimental setup for GPFlow provides concrete evidence of its flexibility in unconditional and conditional generation without requiring a prescribed target length and demonstrates its superior performance in designability and distributional fitness across all tasks.

One limitation of GPFlow is its sensitivity to scheduler choices, especially in multimodal setups where different modalities may interact. Additional sampling techniques such as τ-leaping (Appendix B.5) and localized paths (Appendix C.2.5) may also introduce subtle trade-offs. We are actively exploring empirical training and sampling techniques to develop a more comprehensive unified variable-length generative framework for biological domains.

Acknowledgments

This research is supported in part by the Molecule Maker Lab Institute, an AI Research Institutes program supported by NSF under Award No. 2505932, and the DOE Center for Advanced Bioenergy and Bioproducts Innovation (U.S. Department of Energy, Office of Science, Biological and Environmental Research Program under Award Number DESC0018420). Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the U.S. Department of Energy.

References

[1] Sarah Alamdari, Nitya Thakkar, Rianne van den Berg, Alex X. Lu, Nicolo Fusi, Ava P. Amini, and Kevin K. Yang. Protein generation with evolutionary diffusion: sequence is all you need, September 2023. URL `https://www.biorxiv.org/content/10.1101/2023.09.11.556673v1`. Pages: 2023.09.11.556673 Section: New Results.

[2] Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021.

[3] Minkyung Baek, Frank DiMaio, Ivan Anishchenko, Justas Dauparas, Sergey Ovchinnikov, Gyu Rie Lee, Jue Wang, Qian Cong, Lisa N Kinch, R Dustin Schaeffer, Claudia Millán, Hahnbeom Park, Carson Adams, Caleb R. Glassman, Andy DeGiovanni, Jose H. Pereira, Andria V. Rodrigues, Alberdina A. van Dijk, Ana C. Ebrecht, Diederik J. Opperman, Theo Sagmeister, Christoph Buhlheller, Tea Pavkov-Keller, Manoj K. Rathinaswamy, Udit Dalwadi, Calvin K. Yip, John E. Burke, K. Christopher Garcia, Nick V. Grishin, Paul D. Adams, Randy J. Read, and David Baker. Accurate prediction of protein structures and interactions using a three-track neural network. *Science*, 373(6557):871–876, 2021.

[4] Ethan Baron, Alan N. Amin, Ruben Weitzman, Debora Marks, and Andrew Gordon Wilson. A Diffusion Model to Shrink Proteins While Maintaining Their Function, November 2025. URL `http://arxiv.org/abs/2511.07390`. arXiv:2511.07390 [cs] version: 1.

[5] Helen M Berman, John Westbrook, Zukang Feng, Gary Gilliland, Talapady N Bhat, Helge Weissig, Ilya N Shindyalov, and Philip E Bourne. The protein data bank. *Nucleic acids research*, 28(1):235–242, 2000.

[6] Lukas Billera, Hedwig Nora Nordlinder, Jack Collier Ryder, Anton Oresten, Aron Stålmarck, Theodor Mosetti Björk, and Ben Murrell. Branching flows: Discrete, continuous, and manifold flow matching with splits and deletions. *arXiv preprint arXiv:2511.09465*, 2025.

[7] Pierre Brémaud. An extension of watanabe's theorem of characterization of poisson processes over the positive real half line. *Journal of Applied Probability*, 12(2):396–399, 1975.

[8] Pierre Brémaud. Point processes and queues. *Springer*, 1981.

[9] Jasper Butcher, Rohith Krishna, Raktim Mitra, Rafael I Brent, Yanjing Li, Nathaniel Corley, Paul T Kim, Jonathan Funk, Simon Mathis, Saman Salike, Aiko Muraishi, Helen Eisenach, Tuscan Rock Thompson, Jie Chen, Yuliya Politanska, Enisha Sehgal, Brian Coventry, Odin Zhang, Bo Qiang, Kieran Didi, Max Kazman, Frank DiMaio, and David Baker. De novo design of all-atom biomolecular interactions with rfdiffusion3. *bioRxiv*, 2025.

[10] Andrew Campbell, William Harvey, Christian Weilbach, Valentin De Bortoli, Thomas Rainforth, and Arnaud Doucet. Trans-dimensional generative modeling via jump diffusion models. *Advances in Neural Information Processing Systems*, 36:42217–42257, 2023.

[11] Sidhartha Chaudhury, Sergey Lyskov, and Jeffrey J Gray. Pyrosetta: a script-based interface for implementing molecular modeling algorithms using rosetta. *Bioinformatics*, 26(5):689–691, 2010.

[12] Ricky TQ Chen and Yaron Lipman. Flow matching on general geometries. In *International Conference on Learning Representations*, volume 2024, pages 47922–47945, 2024.

[13] Chaoran Cheng, Jiahan Li, Jian Peng, and Ge Liu. Categorical flow matching on statistical manifolds. *Advances in Neural Information Processing Systems*, 37:54787–54819, 2024.

[14] Chaoran Cheng, Jiahan Li, Jiajun Fan, and Ge Liu. α-flow: A unified framework for continuous-state discrete flow matching models. *arXiv preprint arXiv:2504.10283*, 2025.

[15] Daryl J Daley and David Vere-Jones. *An introduction to the theory of point processes: volume I: elementary theory and methods*. Springer, 2003.

[16] Justas Dauparas, Ivan Anishchenko, Nathaniel Bennett, Hua Bai, Robert J Ragotte, Lukas F Milles, Basile IM Wicky, Alexis Courbet, Rob J de Haas, Neville Bethel, P. J. Y. Leung, T. F. Huddy, S. Pellock, D. Tischer, F. Chan, B. Koepnick, H. Nguyen, A. Kang, B. Sankaran, A. K. Bera, N. P. King, and D. Baker. Robust deep learning–based protein sequence design using proteinmpnn. *Science*, 378(6615):49–56, 2022.

[17] Rohit Dilip, Ayush Varshney, and David Van Valen. Adaptive protein tokenization. *arXiv preprint arXiv:2602.06418*, 2026.

[18] Ian Dunn and David Ryan Koes. Mixed continuous and categorical flow matching for 3d de novo molecule generation. *ArXiv*, pages arXiv–2404, 2024.

[19] Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky TQ Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. Discrete flow matching. *Advances in Neural Information Processing Systems*, 37:133345–133385, 2024.

[20] Benoit Gaujac, Jérémie Donà, Liviu Copoiu, Timothy Atkinson, Thomas Pierrot, and Thomas D Barrett. Learning the language of protein structure. *arXiv preprint arXiv:2405.15840*, 2024.

[21] Tomas Geffner, Kieran Didi, Zuobai Zhang, Danny Reidenbach, Zhonglin Cao, Jason Yim, Mario Geiger, Christian Dallago, Emine Kucukbenli, Arash Vahdat, and Karsten Kreis. Proteina: Scaling flow-based protein structure generative models. *arXiv preprint arXiv:2503.00710*, 2025.

[22] Daniel T Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *The Journal of chemical physics*, 115(4):1716–1733, 2001.

[23] Marton Havasi, Brian Karrer, Itai Gat, and Ricky TQ Chen. Edit flows: Flow matching with edit operations. *arXiv preprint arXiv:2506.09018*, 2025.

[24] Thomas Hayes, Roshan Rao, Halil Akin, Nicholas J. Sofroniew, Deniz Oktay, Zeming Lin, Robert Verkuil, Vincent Q. Tran, Jonathan Deaton, Marius Wiggert, Rohil Badkundri, Irhum Shafkat, Jun Gong, Alexander Derry, Raul S. Molina, Neil Thomas, Yousuf A. Khan, Chetan Mishra, Carolyn Kim, Liam J. Bartie, Matthew Nemeth, Patrick D. Hsu, Tom Sercu, Salvatore Candido, and Alexander Rives. Simulating 500 million years of evolution with a language model. *Science*, 387(6736):850–858, February 2025. doi: 10.1126/science.ads0018. URL `https://www.science.org/doi/10.1126/science.ads0018`. Publisher: American Association for the Advancement of Science.

[25] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[26] Peter Holderrieth, Marton Havasi, Jason Yim, Neta Shaul, Itai Gat, Tommi Jaakkola, Brian Karrer, Ricky TQ Chen, and Yaron Lipman. Generator matching: Generative modeling with arbitrary markov processes. *arXiv preprint arXiv:2410.20587*, 2024.

[27] Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances in neural information processing systems*, 34:12454–12465, 2021.

[28] Chloe Hsu, Robert Verkuil, Jason Liu, Zeming Lin, Brian Hie, Tom Sercu, Adam Lerer, and Alexander Rives. Learning inverse folding from millions of predicted structures. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 8946–8970. PMLR, 2022. URL `https://proceedings.mlr.press/v162/hsu22a.html`.

[29] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589, 2021.

[30] Wolfgang Kabsch and Christian Sander. Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22(12):2577–2637, 1983. doi: 10.1002/bip.360221211.

[31] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. URL `http://arxiv.org/abs/1412.6980`. arXiv:1412.6980 [cs].

[32] Xiangzhe Kong, Rui Jiao, Haowei Lin, Ruihan Guo, Wenbing Huang, Wei-Ying Ma, Zihua Wang, Yang Liu, and Jianzhu Ma. Peptide design through binding interface mimicry with PepMimic. *Nature Biomedical Engineering*, 10:1295–1310, 2026. doi: 10.1038/s41551-025-01507-4. URL `https://doi.org/10.1038/s41551-025-01507-4`.

[33] Patrick Kunzmann and Kay Hamacher. Biotite: a unifying open source computational biology framework in Python. *BMC Bioinformatics*, 19(1):346, 2018. doi: 10.1186/s12859-018-2367-z.

[34] Jiahan Li, Chaoran Cheng, Zuofan Wu, Ruihan Guo, Shitong Luo, Zhizhou Ren, Jian Peng, and Jianzhu Ma. Full-atom peptide design based on multi-modal flow matching. *arXiv preprint arXiv:2406.00735*, 2024.

[35] Jiahan Li, Tong Chen, Shitong Luo, Chaoran Cheng, Jiaqi Guan, Ruihan Guo, Sheng Wang, Ge Liu, Jian Peng, and Jianzhu Ma. Hotspot-driven peptide design via multi-fragment autoregressive extension. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://proceedings.iclr.cc/paper_files/paper/2025/hash/1a0672689a693e0764f93f900488b3d9-Abstract-Conference.html`.

[36] Tiejun Li. Analysis of explicit tau-leaping schemes for simulating chemically reacting systems. *Multiscale Modeling & Simulation*, 6(2):417–436, 2007.

[37] Yeqing Lin, Minji Lee, Zhao Zhang, and Mohammed AlQuraishi. Out of many, one: Designing and scaffolding proteins at the scale of the structural universe with genie 2. *arXiv preprint arXiv:2405.15489*, 2024.

[38] Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, Robert Verkuil, Ori Kabeli, Yaniv Shmueli, Allan dos Santos Costa, Maryam Fazel-Zarandi, Tom Sercu, Salvatore Candido, and Alexander Rives. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637):1123–1130, 2023. doi: 10.1126/science.ade2574.

[39] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

[40] Sidney Lyayuga Lisanza, Jacob Merle Gershon, Samuel WK Tipps, Jeremiah Nelson Sims, Lucas Arnoldt, Samuel J Hendel, Miriam K Simma, Ge Liu, Muna Yase, Hongwei Wu, Claire D. Tharp, Xinting Li, Alex Kang, Evans Brackenbrough, Asim K. Bera, Stacey Gerben, Bruce J. Wittmann, Andrew C. McShan, and David Baker. Multistate and functional protein design using rosettafold sequence space diffusion. *Nature biotechnology*, 43(8):1288–1298, 2025.

[41] Nir London, Barak Raveh, Eyal Cohen, Guy Fathi, and Ora Schueler-Furman. Rosetta flexpepdock web server—high resolution modeling of peptide–protein interactions. *Nucleic acids research*, 39(suppl_2):W249–W253, 2011.

[42] Valerio Mariani, Marco Biasini, Alessandro Barbato, and Torsten Schwede. lDDT: A local superposition-free score for comparing protein structures and models using distance difference tests. *Bioinformatics*, 29(21):2722–2728, 2013. doi: 10.1093/bioinformatics/btt473.

[43] John Nguyen, Marton Havasi, Tariq Berrada, Luke Zettlemoyer, and Ricky TQ Chen. Oneflow: Concurrent mixed-modal and interleaved generation with edit flows. *arXiv preprint arXiv:2510.03506*, 2025.

[44] William Peebles and Saining Xie. Scalable Diffusion Models with Transformers, March 2023. URL http://arxiv.org/abs/2212.09748. arXiv:2212.09748 [cs].

[45] Xingang Peng, Ruihan Guo, Fenglin Guo, Ziyi Wang, Jiayu Sun, Jiaqi Guan, Yinjun Jia, Yan Xu, Yanwen Huang, Muhan Zhang, Jian Peng, Xinquan Wang, Chuanhui Han, Zihua Wang, and Jianzhu Ma. Unified modeling of 3D molecular generation via atomic interactions with PocketXMol. *Cell*, 189(7):1904–1922.e28, 2026. doi: 10.1016/j.cell.2026.01.003. URL https://doi.org/10.1016/j.cell.2026.01.003.

[46] Jiezhong Qiu, Junde Xu, Jie Hu, Hanqun Cao, Liya Hou, Zijun Gao, Xinyi Zhou, Anni Li, Xiujuan Li, Bin Cui, Fei Yang, Shuang Peng, Ning Sun, Fangyu Wang, Aimin Pan, Jie Tang, Jieping Ye, Junyang Lin, Jin Tang, Xingxu Huang, Pheng Ann Heng, and Guangyong Chen. Instructplm: Aligning protein language models to follow protein structure instructions. *bioRxiv*, pages 2024–04, 2024.

[47] Yanru Qu, Cheng-Yen Hsieh, Zaixiang Zheng, Ge Liu, and Quanquan Gu. Protein autoregressive modeling via multiscale structure generation. *arXiv preprint arXiv:2602.04883*, 2026.

[48] Barak Raveh, Nir London, Lior Zimmerman, and Ora Schueler-Furman. Rosetta flexpepdock ab-initio: simultaneous folding, docking and refinement of peptides onto their receptors. *PloS one*, 6(4):e18934, 2011.

[49] Yinuo Ren, Haoxuan Chen, Yuchen Zhu, Wei Guo, Yongxin Chen, Grant M Rotskoff, Molei Tao, and Lexing Ying. Fast solvers for discrete diffusion models: Theory and applications of high-order algorithms. *arXiv preprint arXiv:2502.00234*, 2025.

[50] Subham Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.

[51] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

[52] Martin Steinegger and Johannes Söding. Mmseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature biotechnology*, 35(11):1026–1028, 2017.

[53] Baris E. Suzek, Hongzhan Huang, Peter McGarvey, Raja Mazumder, and Cathy H. Wu. UniRef: comprehensive and non-redundant UniProt reference clusters. *Bioinformatics*, 23(10):1282–1288, May 2007. ISSN 1367-4803. doi: 10.1093/bioinformatics/btm098. URL https://doi.org/10.1093/bioinformatics/btm098.

[54] The UniProt Consortium. UniProt: the Universal Protein Knowledgebase in 2023. *Nucleic Acids Research*, 51(D1):D523–D531, January 2023. ISSN 0305-1048. doi: 10.1093/nar/gkac1052. URL https://doi.org/10.1093/nar/gkac1052.

[55] Michel Van Kempen, Stephanie S Kim, Charlotte Tumescheit, Milot Mirdita, Jeongjae Lee, Cameron LM Gilchrist, Johannes Söding, and Martin Steinegger. Fast and accurate protein structure search with foldseek. *Nature biotechnology*, 42(2):243–246, 2024.

[56] Lei Wang, Xudong Li, Han Zhang, Jinyi Wang, Dingkang Jiang, Zhidong Xue, and Yan Wang. A comprehensive review of protein language models. *arXiv preprint arXiv:2502.06881*, 2025.

[57] Xinyou Wang, Zaixiang Zheng, Fei Ye, Dongyu Xue, Shujian Huang, and Quanquan Gu. Diffusion Language Models Are Versatile Protein Learners, October 2024. URL http://arxiv.org/abs/2402.18567. arXiv:2402.18567 [cs].

[58] 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, Basile I. M. Wicky, Nikita Hanikel, Samuel J. Pellock, Alexis Courbet, William Sheffler, Jue Wang, Preetham

Venkatesh, Isaac Sappington, Susana Vázquez Torres, Anna Lauko, Valentin De Bortoli, Emile Mathieu, Sergey Ovchinnikov, Regina Barzilay, Tommi S. Jaakkola, Frank DiMaio, Minkyung Baek, and David Baker. De novo design of protein structure and function with rfdiffusion. *Nature*, 620(7976):1089–1100, 2023.

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

[60] 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*, 2023.

[61] Yang Zhang and Jeffrey Skolnick. Scoring function for automated assessment of protein structure template quality. *Proteins: Structure, Function, and Bioinformatics*, 57(4):702–710, 2004. doi: 10.1002/prot.20264.

[62] Yang Zhang and Jeffrey Skolnick. Tm-align: a protein structure alignment algorithm based on the tm-score. *Nucleic acids research*, 33(7):2302–2309, 2005.

Supplementary Material

Contents

A Theoretical Details **17**

A.1 Proof for Theorem 3.1 (Conditional Rate) 17

A.2 Proof for Theorem 3.2 (Marginal Rate) 17

A.3 Proposition A.1 (Poisson NLL) 18

A.4 Proof for Theorem 3.3 (Multimodal GPFlow) 19

A.5 Theorem A.2 (Order-Preserving GPFlow) 20

A.6 Proof for Theorem 4.1 (KL Bound) 21

A.7 Generator for GPFlow 23

A.8 Connection to EditFlow 23

B Algorithmic Details **24**

B.1 Order-Preserving Generalized Poisson Flow 24

B.2 Training and Sampling Pseudocode 26

B.3 Flow Parameterization 27

B.4 Sampling Distribution Parameterization 29

B.5 τ-Leaping 29

B.6 Additional Related Work 30

C Experiment Details **30**

C.1 Protein Structure Generation 31

C.2 Protein Sequence Generation 34

C.3 Peptide Co-Design 37

D Ablation Studies **39**

D.1 Insertion Scheduler and Sampling Step Ablation 39

D.2 Designability-Diversity Trade-Off 40

D.3 τ-Leaping Ablation 40

D.4 Sequence Generation Technique Ablation 40

E Additional Results **41**

E.1 Unconditional Protein Generation 41

E.2 Sequence-Based Length Control 41

E.3 Structure-Based Motif Scaffolding 42

E.4 Sequence-Based Motif Scaffolding 42

E.5 Peptide Co-Design 43

F Broader Impacts **43**

A Theoretical Details

In this section, we provide proofs for the major theorems established in the main context. We also discuss the technical details of interpreting GPFlow as a generator and its connections to existing models such as EditFlow.

A.1 Proof for Theorem 3.1 (Conditional Rate)

Theorem 3.1. *Let $\kappa : [0,1] \to [0,1]$ be a nondecreasing, absolutely continuous scheduler with $\kappa_0 = 0$ and $\kappa_1 = 1$, and define the* completion time $\tau_\kappa := \inf\{t : \kappa_t = 1\}$. *For any fixed $L \geq 1$, assume $\kappa_t < 1$ for $t < \tau_\kappa$ and define the insertion* hazard *and* conditional rate *by*

$$h_t := \begin{cases} \dfrac{\dot{\kappa}_t}{1-\kappa_t}, & t < \tau_\kappa, \\ 0, & t \geq \tau_\kappa, \end{cases} \qquad \lambda_t(X_t|L) := (L - X_t)h_t. \tag{1}$$

Then, the resulting process satisfies $X_t \sim B(L, \kappa_t)$ for $t < \tau_\kappa$ and is absorbed at $X_t = L$ for $t \geq \tau_\kappa$.

Proof. Consider $Y_t = L - X_t$ as a pure-death process with death rate $\mu_t(Y_t) = Y_t h_t$. Let $p_t(k) := \Pr(Y_t = k)$. The Kolmogorov forward equation reads:

$$\dot{p}_t(k) = \mu_t(k+1)p_t(k+1) - \mu_t(k)p_t(k) = h_t\left[(k+1)p_t(k+1) - kp_t(k)\right]. \tag{13}$$

Consider the probability generating function (PGF) $G(z,t) := \mathbb{E}[z^{Y_t}] = \sum_{k=0}^{L} p_t(k)z^k$. We have

$$\frac{\partial G}{\partial t} = \sum_{k=0}^{L} \dot{p}_t(k)z^k = h_t \sum_{k=0}^{L} \left[(k+1)p_t(k+1) - kp_t(k)\right] z^k \tag{14}$$

$$= h_t\left(\frac{\partial G}{\partial z} - z\frac{\partial G}{\partial z}\right) = h_t(1-z)\frac{\partial G}{\partial z}. \tag{15}$$

One can verify that the function

$$G(z,t) = (\kappa_t + z(1-\kappa_t))^L \tag{16}$$

satisfies the above PDE with the initial condition $Y_0 = L$. As $G(z,t)$ is the standard PGF for the binomial distribution $B(L, 1-\kappa_t)$, we have $Y_t \sim B(L, 1-\kappa_t)$ and $X_t \sim B(L, \kappa_t)$ for $t < \tau_\kappa$. As $t \to \tau_\kappa^-$, $\kappa_t \to 1^-$ and $\mathbb{E}[Y_t] = L(1-\kappa_t) \to 0$.

Moreover, $Y_t = L - X_t$ is nonnegative and nonincreasing along every pure-birth path, so its pathwise limit exists; the vanishing expectation implies that this limit is zero almost surely. Hence $X_t \to L$ almost surely. Defining L as absorbing and setting the rate to zero for $t \geq \tau_\kappa$ yields the stated result without evaluating the hazard after saturation. □

The general definition of the scheduler above allows us to use additional scheduler families without compromising theoretical validity. In particular, the early-insertion schedule $\kappa_t = \min\{1, t/\tau\}$ has $h_t = 1/(\tau - t)$ for $t < \tau$; at $t = \tau$, the process has reached L almost surely, after which the state is absorbing, and the rate is defined to be zero. Thus, the expression $\dot{\kappa}_t/(1-\kappa_t)$ is never evaluated in the saturated regime. In the losses, the target insertion contribution is also set to zero for $t \geq \tau_\kappa$. Such a scheduler was already used in prior work, such as TDDM [10], but it was never rigorously formulated.

A.2 Proof for Theorem 3.2 (Marginal Rate)

Theorem 3.2. *For any scheduler κ_t defined above, let q be a target length distribution, define $p_t(k|y) := \Pr(X_t = k|X_1 = y)$, and let $p_t(k) := \sum_y p_t(k|y)q(y)$. For every k with $p_t(k) > 0$, the posterior probability is $p(X_1 = y|X_t = k) := p_t(k|y)q(y)/p_t(k)$. Then the* marginal rate

$$\lambda_t^*(k) := \mathbb{E}_{X_1 \sim p(\cdot|X_t=k)}[\lambda_t(k|X_1)] \tag{2}$$

generates the marginal path p_t and hence $p_1 = q$.

Proof. The Kolmogorov forward equation reads:

$$\dot{p}_t(k|y) = \lambda_t(k-1|y)p_t(k-1|y) - \lambda_t(k|y)p_t(k|y). \tag{17}$$

For the marginal $p_t(k) := \sum_y q(y)p_t(k|y)$, differentiating with respect to time, we obtain:

$$\dot{p}_t(k) = \sum_y q(y)\dot{p}_t(k|y). \tag{18}$$

Substituting the Kolmogorov equation, we get

$$\dot{p}_t(k) = \underbrace{\sum_y q(y)\lambda_t(k-1|y)p_t(k-1|y)}_{\text{event}} - \underbrace{\sum_y q(y)\lambda_t(k|y)p_t(k|y)}_{\text{survival}}. \tag{19}$$

For every k with $p_t(k) > 0$, Bayes' rule gives the posterior probability stated in Theorem 3.2:

$$p(X_1 = y|X_t = k) = \frac{p_t(k|y)q(y)}{p_t(k)}. \tag{20}$$

Therefore, the marginal rate conditioned on the explicit current state $X_t = k$ is

$$\lambda_t^*(k) := \mathbb{E}_{X_1 \sim p(\cdot|X_t=k)}[\lambda_t(k|X_1)] = \sum_y \frac{p_t(k|y)q(y)}{p_t(k)}\lambda_t(k|y). \tag{21}$$

Therefore,

$$\lambda_t^*(k)p_t(k) = \sum_y q(y)p_t(k|y)\lambda_t(k|y), \tag{22}$$

which is the survival term in Eq. 19. Similarly, we have

$$\lambda_t^*(k-1)p_t(k-1) = \sum_y q(y)p_t(k-1|y)\lambda_t(k-1|y), \tag{23}$$

which is the event term in Eq. 19. Combining both results:

$$\dot{p}_t(k) = \lambda_t^*(k-1)p_t(k-1) - \lambda_t^*(k)p_t(k), \tag{24}$$

which is precisely the Kolmogorov forward equation for a generalized Poisson process driven by the rate $\lambda_t^*(X_t)$. Since $p_1(k|y) = \mathbb{1}\{k = y\}$, the terminal marginal is $p_1(k) = q(k)$. □

A.3 Proposition A.1 (Poisson NLL)

Proposition A.1. *For a target generalized Poisson process with rate function λ_t^*, consider a realization on $[0, 1]$ with event times $\{t_k\}_{k=1}^L$, where $0 < t_k \leq 1$. Its negative log-likelihood (NLL) under an estimated rate λ_t is:*

$$\mathcal{L}_{GP} = \int_0^1 \lambda_t \, \mathrm{d}t - \sum_{k=1}^L \log \lambda_{t_k}. \tag{25}$$

Proof. At each time step t, the probability of an event is $\lambda_t \, \mathrm{d}t$, whereas the probability of survival is $1 - \lambda_t \, \mathrm{d}t$. Therefore, we have

$$\mathcal{L}_{\text{GP}} = -\log\left(\prod_{t \in \text{no event}} (1 - \lambda_t \mathrm{d}t) \cdot \prod_{k=1}^L \lambda_{t_k} \mathrm{d}t\right) \tag{26}$$

$$= -\log\left(\prod_{t=0}^1 (1 - \lambda_t \mathrm{d}t) \cdot \prod_{k=1}^L \lambda_{t_k}\right) \tag{27}$$

$$= \int_0^1 \lambda_t \mathrm{d}t - \sum_{k=1}^L \log \lambda_{t_k}, \tag{28}$$

which concludes the proof. □

The identity used in Section 3.1 to rewrite the event term of the Poisson NLL follows from the integration (smoothing) formula for the stochastic intensity of a point process [8]. Let N_t be a simple point process with event times $\{t_k\}$, adapted to a filtration $\{\mathcal{F}_t\}$, that admits an $\mathcal{F}_t$-stochastic intensity λ_t^*, i.e., $N_t - \int_0^t \lambda_s^* \, \mathrm{d}s$ is an $\mathcal{F}_t$-martingale. Then, for every nonnegative $\mathcal{F}_t$-predictable process H_t:

$$\mathbb{E}\left[\sum_k H_{t_k}\right] = \mathbb{E}\left[\int_0^1 H_t \mathrm{d}N_t\right] = \mathbb{E}\left[\int_0^1 H_t \lambda_t^* \, \mathrm{d}t\right], \tag{29}$$

a property that in fact characterizes the stochastic intensity λ_t^*; the deterministic-integrand case recovers Campbell's theorem [15]. By Theorem 3.2, the marginal rate λ_t^* is such a stochastic intensity (the process conditioned on $X_1 = L$ is an inhomogeneous pure-birth process with predictable compensator [7]). Choosing the predictable integrand $H_t = -\log \lambda_t(X_{t^-})$ with the learned rate λ_t then yields:

$$\mathbb{E}_{\{t_k\},L}\left[-\sum_k \log \lambda_{t_k}\right] = \mathbb{E}\left[-\int_0^1 \lambda_t^* \log \lambda_t \, \mathrm{d}t\right] = \mathbb{E}_t[-\lambda_t^* \log \lambda_t], \tag{30}$$

which is the identity in the main text. In this way, the NLL loss objective is fully simulation-free.

A.4 Proof for Theorem 3.3 (Multimodal GPFlow)

We first describe a measure-theoretic characterization of the variable-length state space Y that ensures that defining a probability path is mathematically well-posed. Let $(\mathsf{S}, \mathcal{S}, \nu)$ be a component space with a σ-finite reference measure. We use Lebesgue measure for Euclidean components, counting measure for categorical components, Riemannian volume for manifold-valued components, and the corresponding product measure for mixed components. Define

$$\mathsf{Y} := \bigsqcup_{k \geq 0} \{k\} \times \mathsf{S}^k, \qquad \mu := \sum_{k \geq 0} \delta_k \otimes \nu^{\otimes k}. \tag{31}$$

Thus, a probability law on variable-length objects can be represented by a density or mass function with respect to the single reference measure μ.

For $y = (k, s) \in \mathsf{Y}$, let $\mathsf{M}(y)$ be the insertion-variable space and let $I(y, m) \in \{k+1\} \times \mathsf{S}^{k+1}$ be the measurable insertion map. In the order-agnostic case, m contains only the new component value; in the ordered case, it also contains an insertion slot. For a clean target $Y_1 = y_1$, let $\rho_t(\mathrm{d}m|y, y_1)$ be a probability kernel on $\mathsf{M}(y)$. The associated conditional jump measure is

$$Q_t(A|y, y_1) := \lambda_t(y|y_1) \int_{\mathsf{M}(y)} \mathbb{1}\{I(y, m) \in A\} \rho_t(\mathrm{d}m|y, y_1). \tag{32}$$

This equation makes the coupling explicit: an insertion value is drawn from $\rho_t(\cdot|y, y_1)$ only when an event governed by $\lambda_t(y|y_1)$ occurs. The total mass of $Q_t(\cdot|y, y_1)$ is exactly $\lambda_t(y|y_1)$.

Theorem 3.3. *Let $Y_1 = (X_1, S_1) \sim q$ index conditional probability paths $p_t(\cdot|Y_1)$ on Y, with $p_1(\cdot|Y_1) = \delta_{Y_1}$. At state $y \in \mathsf{Y}$, suppose each conditional process has within-length generator $\mathcal{B}_t^{Y_1}$, insertion rate $\lambda_t(y|Y_1)$, and sampling distribution $\rho_t(\mathrm{d}m|y, Y_1)$ for the insertion variable m. For any y with $p_t(y) > 0$, the posterior of the clean target is $p(\mathrm{d}y_1|Y_t = y) := p_t(y|y_1)q(\mathrm{d}y_1)/p_t(y)$, where $p_t = \int p_t(\cdot|y_1)q(\mathrm{d}y_1)$. Define the marginals by expectations over this posterior:*

$$\mathcal{B}_t^* f(y) := \mathbb{E}_{Y_1 \sim p(\cdot|Y_t = y)}[\mathcal{B}_t^{Y_1} f(y)], \quad \lambda_t^*(y) := \mathbb{E}_{Y_1 \sim p(\cdot|Y_t = y)}[\lambda_t(y|Y_1)], \tag{5}$$

$$\rho_t^*(\mathrm{d}m|y) := \frac{\mathbb{E}_{Y_1 \sim p(\cdot|Y_t = y)}[\lambda_t(y|Y_1)\rho_t(\mathrm{d}m|y, Y_1)]}{\mathbb{E}_{Y_1 \sim p(\cdot|Y_t = y)}[\lambda_t(y|Y_1)]}, \tag{6}$$

where ρ_t^* can be arbitrary when the marginal rate is zero. Then the process with $\mathcal{B}_t^*$ and insertion kernel $Q_t^* = \lambda_t^* \rho_t^*$ generates the marginal p_t and hence $p_1 = q$.

Proof. This proof follows the outline in transdimensional jump diffusion [10], while extending it to allow the inserted component to be continuous, discrete, Riemannian, or mixed. Let $\mathcal{B}_t^{y_1}$ denote the

conditional within-length generator. For every bounded test function f in the domain of the generator, the conditional path satisfies the weak forward equation:

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{\mathsf{Y}} f(y)p_t(\mathrm{d}y|y_1) = \int_{\mathsf{Y}}\left[\mathcal{B}_t^{y_1}f(y) + \int_{\mathsf{Y}}(f(y')-f(y))Q_t(\mathrm{d}y'|y,y_1)\right]p_t(\mathrm{d}y|y_1). \tag{33}$$

For brevity in the proof, denote the posterior law written explicitly in Theorem 3.3 by

$$\pi_t(\mathrm{d}y_1|y) := p(\mathrm{d}y_1|Y_t = y). \tag{34}$$

Equivalently, it is characterized by the disintegration identity $q(\mathrm{d}y_1)p_t(\mathrm{d}y|y_1) = p_t(\mathrm{d}y)\pi_t(\mathrm{d}y_1|y)$. Thus, the three posterior expectations in the theorem are:

$$\mathcal{B}_t^* f(y) = \int \mathcal{B}_t^{y_1} f(y)\pi_t(\mathrm{d}y_1|y), \tag{35}$$

$$\lambda_t^*(y) = \int \lambda_t(y|y_1)\pi_t(\mathrm{d}y_1|y), \tag{36}$$

$$\rho_t^*(\mathrm{d}m|y) = \frac{\int \lambda_t(y|y_1)\rho_t(\mathrm{d}m|y,y_1)\pi_t(\mathrm{d}y_1|y)}{\int \lambda_t(y|y_1)\pi_t(\mathrm{d}y_1|y)}. \tag{37}$$

Integrating Eq. 33 over $q(\mathrm{d}y_1)$ and applying the disintegration identity then yields the marginal jump measure

$$Q_t^*(A|y) := \int Q_t(A|y,y_1)\pi_t(\mathrm{d}y_1|y) = \lambda_t^*(y)\int_{\mathsf{M}(y)} \mathbb{1}\{I(y,m)\in A\}\rho_t^*(\mathrm{d}m|y), \tag{38}$$

where λ_t^* and ρ_t^* are precisely those in Theorem 3.3. Therefore,

$$\frac{\mathrm{d}}{\mathrm{d}t}\int f(y)p_t(\mathrm{d}y) = \int\left[\mathcal{B}_t^* f(y) + \int (f(y')-f(y))Q_t^*(\mathrm{d}y'|y)\right]p_t(\mathrm{d}y), \tag{39}$$

which is the weak forward equation of the claimed marginal process. Since $p_1(\cdot|y_1) = \delta_{y_1}$ for q-almost every y_1, its endpoint is $p_1 = q$. □

For intuition, the corresponding forward equation can be written, as an identity of measures, as

$$\partial_t p_t = \mathcal{B}_t^{*\dagger} p_t - \lambda_t^* p_t + \int_{\mathsf{Y}} p_t(\mathrm{d}\bar{y})Q_t^*(\cdot|\bar{y}). \tag{40}$$

The second term is outgoing probability mass, whereas the third term is inflow from predecessor states; only the outgoing insertion measure $Q_t^* = \lambda_t^*\rho_t^*$ belongs to the generator at the current state. When S is continuous, $\mathcal{B}_t^{*\dagger}p_t = -\nabla\cdot(v_t^* p_t)$ (or its diffusion/Riemannian analog), and integrals over inserted values are ordinary integrals. When S is discrete, the same display is a master equation, all integrals over S become sums, and $\mathcal{B}_t^*$ is a within-length CTMC generator. Hence, the proof applies to both cases without treating a categorical component as if it had a Euclidean density.

A.5 Theorem A.2 (Order-Preserving GPFlow)

In the order-preserving specialization, $\mathsf{M}(y) = \{0,\dots,k\}\times\mathsf{S}$ and $m = (i,a)$. The general insertion map becomes $I_i(y,a) := I(y,m) = I(y,(i,a))$, where I_i places component a in slot i. Thus, the slot is part of the insertion variable whose rate-weighted distribution is marginalized; the insertion map itself remains deterministic. A detailed construction is provided in Appendix B.1.

Theorem A.2. *At a state $Y_t = y = (k,s)$, let the insertion variable be $m = (i,a)$, where $i \in \{0,\dots,k\}$ is an insertion slot and $a \in \mathsf{S}$ is the new component. Using the posterior law $p(\mathrm{d}y_1|Y_t = y)$ from Theorem 3.3, define for each slot*

$$(\lambda_t^i)^*(y) := \mathbb{E}_{Y_1\sim p(\cdot|Y_t=y)}[\lambda_t^i(y|Y_1)], \tag{41}$$

$$(\rho_t^i)^*(\mathrm{d}a|y) := \frac{\mathbb{E}_{Y_1\sim p(\cdot|Y_t=y)}[\lambda_t^i(y|Y_1)\rho_t^i(\mathrm{d}a|y,Y_1)]}{\mathbb{E}_{Y_1\sim p(\cdot|Y_t=y)}[\lambda_t^i(y|Y_1)]}. \tag{42}$$

Together with the marginal within-length generator $\mathcal{B}_t^*$ from Theorem 3.3, the insertion jump measure

$$Q_t^*(\mathrm{d}y'|y) = \sum_{i=0}^{k}(\lambda_t^i)^*(y)\int_{\mathsf{S}}\delta_{I_i(y,a)}(\mathrm{d}y')(\rho_t^i)^*(\mathrm{d}a|y) \tag{43}$$

generates the ordered target distribution $Y_1 = (X_1,S_1)\sim q$, where $S_1 = (S_1^1,\dots,S_1^{X_1})$. As before, $(\rho_t^i)^*$ may be chosen arbitrarily if $(\lambda_t^i)^*(y) = 0$.

Proof. The order-preserving process is the specialization of Theorem 3.3 in which the insertion variable is $m = (i, a)$ and $I(y, (i, a)) = I_i(y, a)$. Its outgoing insertion jump measure is therefore:

$$Q_t^*(\mathrm{d}y'|y) = \sum_{i=0}^{k} (\lambda_t^i)^*(y) \int_{\mathsf{S}} \delta_{I_i(y,a)}(\mathrm{d}y')(\rho_t^i)^*(\mathrm{d}a|y). \tag{44}$$

For a clean target of length L and current length k, the thinning path partitions the $L - k$ remaining target components into insertion bins $\Omega_0, \ldots, \Omega_k$ (see Appendix B.1 for construction). The conditional slot rate is $|\Omega_i| h_t$, hence the total conditional rate is:

$$\sum_{i=0}^{k} \lambda_t^i(k|L) = (L - k)h_t, \tag{45}$$

which recovers the conditional rate in Theorem 3.1. Posterior marginalization of each slot-specific insertion measure gives exactly the rates and insertion kernels in Theorem A.2. Therefore, the terminal distribution is the ordered target distribution q. □

The general weak-forward argument from Appendix A.4 then yields

$$\partial_t p_t = \mathcal{B}_t^{*\dagger} p_t - p_t \sum_{i=0}^{k} (\lambda_t^i)^* + \int_{\mathsf{Y}} p_t(\mathrm{d}\bar{y}) Q_t^*(\cdot|\bar{y}), \tag{46}$$

and the insertion maps I_i preserve the clean-target ordering by construction.

A.6 Proof for Theorem 4.1 (KL Bound)

Theorem 4.1. *Let $\{X_t\}_{t\geq 0}$ and $\{\tilde{X}_t\}_{t\geq 0}$ be two Markov processes with generators $\mathcal{A}_t$ and $\tilde{\mathcal{A}}_t$, respectively. Suppose $\mathcal{A}_t$ is determined by (u_t, σ_t^2, Q_t) and $\tilde{\mathcal{A}}_t$ is determined by $(\tilde{u}_t, \sigma_t^2, \tilde{Q}_t)$, so they coincide in the diffusion coefficient. Assume that both processes have the same initial distribution p_0 and that sufficient regularity conditions hold. Then, the KL divergence between p_T and $\tilde{p}_T$ can be bounded by the loss terms $\mathcal{L}_t(x)$ as*

$$\mathcal{D}_{KL}(p_T \| \tilde{p}_T) \leq \int_0^T \mathbb{E}_{x\sim p_t} \left[\mathcal{L}_t(x)\right] \mathrm{d}t, \quad \mathcal{L}_t(x) = \frac{\|u_t(x) - \tilde{u}_t(x)\|_2^2}{2\sigma_t^2} + \mathcal{D}_{KL}(Q_t \| \tilde{Q}_t), \tag{10}$$

where the first drift-diffusion term is omitted when no continuous component is present, and $\mathcal{D}_{KL}(Q_t \| \tilde{Q}_t)$ denotes the generalized KL divergence between two positive measures:

$$\mathcal{D}_{KL}(Q \| \tilde{Q}) = \int \tilde{Q}(\mathrm{d}y|x) + \int \left(\log \frac{\mathrm{d}Q(y|x)}{\mathrm{d}\tilde{Q}(y|x)} - 1\right) Q(\mathrm{d}y|x). \tag{11}$$

Proof. We first discuss one sufficient set of regularity conditions. These conditions ensure that the Radon-Nikodym derivative and every expression below are well-defined.

Both processes start from the same distribution p_0. For every $t \in [0, T]$, p_t and $\tilde{p}_t$ are strictly positive densities with respect to a common σ-finite reference measure; they are differentiable in time and twice differentiable in every continuous coordinate. The densities, their derivatives, and all generator terms are integrable so that differentiation may be exchanged with integration, the generator identities may be applied, and the continuous-coordinate integration-by-parts boundary terms vanish. On every continuous component, the two processes have the same measurable diffusion coefficient $\sigma_t^2 I$ with $\sigma_t > 0$, and the time integral of $\mathbb{E}_{p_t}[\|u_t - \tilde{u}_t\|^2/\sigma_t^2]$ is finite. The jump measures are finite kernels, $Q_t(\cdot|x)$ is absolutely continuous with respect to $\tilde{Q}_t(\cdot|x)$ for $\mathrm{d}t\, p_t(\mathrm{d}x)$-almost every (t, x), and the generalized KL term is measurable and integrable over time and p_t. If no continuous component is present, the regularity conditions are unnecessary.

The proof uses the generator identity together with completing the square for the continuous component and the nonnegativity of $z \log z - z + 1$ for the jump component. For readability, we suppress

the time subscript on Q_t and $\tilde{Q}_t$ below. Consider the time derivative of the KL divergence:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{D}_{\mathrm{KL}}(p_t\|\tilde{p}_t) = \frac{\mathrm{d}}{\mathrm{d}t}\int p_t(x)\log\frac{p_t(x)}{\tilde{p}_t(x)}\,\mathrm{d}x \tag{47}$$

$$= \int \dot{p}_t(x)\log\frac{p_t(x)}{\tilde{p}_t(x)}\,\mathrm{d}x + \underbrace{\int p_t(x)\frac{\dot{p}_t(x)}{p_t(x)}\,\mathrm{d}x}_{=0} - \int \frac{\mathrm{d}\tilde{p}_t}{\mathrm{d}t}(x)\frac{p_t(x)}{\tilde{p}_t(x)}\,\mathrm{d}x \tag{48}$$

$$= \mathbb{E}_{x\sim p_t}\left[\mathcal{A}_t\left(\log\frac{p_t}{\tilde{p}_t}\right)(x)\right] - \mathbb{E}_{x\sim\tilde{p}_t}\left[\tilde{\mathcal{A}}_t\left(\frac{p_t}{\tilde{p}_t}\right)(x)\right] \tag{49}$$

$$= \mathbb{E}_{x\sim p_t}\left[\mathcal{A}_t\left(\log\frac{p_t}{\tilde{p}_t}\right)(x) - \frac{\tilde{p}_t(x)}{p_t(x)}\tilde{\mathcal{A}}_t\left(\frac{p_t}{\tilde{p}_t}\right)(x)\right]. \tag{50}$$

Below, for simplicity, we denote $f = p_t/\tilde{p}_t$. Then

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{D}_{\mathrm{KL}}(p_t\|\tilde{p}_t) \tag{51}$$

$$= \frac{\mathrm{d}}{\mathrm{d}t}\int p_t(x)\log\frac{p_t(x)}{\tilde{p}_t(x)}\,\mathrm{d}x = \mathbb{E}_{x\sim p_t}\left[\mathcal{A}_t(\log f)(x) - \frac{1}{f(x)}\tilde{\mathcal{A}}_t f(x)\right] \tag{52}$$

$$= \mathbb{E}_{x\sim p_t}\Bigg[u_t(x)\cdot\nabla\log f(x) + \frac{1}{2}\sigma_t^2\nabla\cdot\nabla\log f(x) + \int[\log f(y) - \log f(x)]Q(\mathrm{d}y|x) \tag{53}$$

$$- \frac{1}{f(x)}\left(\tilde{u}_t(x)\cdot\nabla f(x) + \frac{1}{2}\sigma_t^2\nabla\cdot\nabla f(x) + \int[f(y) - f(x)]\tilde{Q}(\mathrm{d}y|x)\right)\Bigg].$$

Substitute $\nabla f(x)/f(x) = \nabla\log f(x)$ and $\nabla\cdot\nabla f(x)/f(x) = \nabla\cdot\nabla\log f(x) + \|\nabla\log f(x)\|^2$ in Eq. 53 to get:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{D}_{\mathrm{KL}}(p_t\|\tilde{p}_t) \tag{54}$$

$$= \mathbb{E}_{x\sim p_t}\Bigg[u_t(x)\cdot\nabla\log f(x) + \frac{1}{2}\sigma_t^2\nabla\cdot\nabla\log f(x) + \int[\log f(y) - \log f(x)]Q(\mathrm{d}y|x) \tag{55}$$

$$- \tilde{u}_t(x)\cdot\frac{\nabla f(x)}{f(x)} - \frac{1}{2}\sigma_t^2\frac{\nabla\cdot\nabla f(x)}{f(x)} - \int\left(\frac{f(y)}{f(x)} - 1\right)\tilde{Q}(\mathrm{d}y|x)\Bigg]$$

$$= \mathbb{E}_{x\sim p_t}\Bigg[(u_t(x) - \tilde{u}_t(x))\cdot\nabla\log f(x) - \frac{1}{2}\sigma_t^2\|\nabla\log f(x)\|^2 \tag{56}$$

$$+ \int\left(\log\frac{f(y)}{f(x)}\right)Q(\mathrm{d}y|x) + \int\left(1 - \frac{f(y)}{f(x)}\right)\tilde{Q}(\mathrm{d}y|x)\Bigg].$$

Note that

$$(u_t(x) - \tilde{u}_t(x))\cdot\nabla\log f(x) - \frac{\sigma_t^2}{2}\|\nabla\log f(x)\|^2 \tag{57}$$

$$= \frac{\|u_t(x) - \tilde{u}_t(x)\|^2}{2\sigma_t^2} - \frac{\sigma_t^2}{2}\left\|\nabla\log f(x) - \frac{u_t(x) - \tilde{u}_t(x)}{\sigma_t^2}\right\|^2 \le \frac{\|u_t(x) - \tilde{u}_t(x)\|^2}{2\sigma_t^2}, \tag{58}$$

For the jump terms, define $a(y) := f(y)/f(x)$ and $r(y) := \mathrm{d}Q(y|x)/\mathrm{d}\tilde{Q}(y|x)$. The pointwise inequality $r\log a + 1 - a \le r\log r - r + 1$ holds because the difference between the right- and left-hand sides is $a[(r/a)\log(r/a) - r/a + 1] \ge 0$. Consequently,

$$\int\log\frac{f(y)}{f(x)}Q(\mathrm{d}y|x) + \int\left(1 - \frac{f(y)}{f(x)}\right)\tilde{Q}(\mathrm{d}y|x) \tag{59}$$

$$\le \int(r\log r - r + 1)\,\tilde{Q}(\mathrm{d}y|x) = \mathcal{D}_{\mathrm{KL}}(Q(\cdot|x)\|\tilde{Q}(\cdot|x)). \tag{60}$$

Combining the continuous and jump bounds, we conclude:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{D}_{\mathrm{KL}}(p_t\|\tilde{p}_t) \le \mathbb{E}_{x\sim p_t}\left[\frac{\|u_t(x)-\tilde{u}_t(x)\|^2}{2\sigma_t^2} + \int\left(\log\frac{\mathrm{d}Q(y|x)}{\mathrm{d}\tilde{Q}(y|x)} - 1\right)Q(\mathrm{d}y|x) + \int\tilde{Q}(\mathrm{d}y|x)\right], \tag{61}$$

which completes the proof by integration. If no continuous component is present, the completion-of-the-square step and its resulting drift-diffusion term are absent, leaving only the jump-measure term. □

A.7 Generator for GPFlow

Table 4: Generators for each variant of GPFlow.

Model Variant	Length-Only	Order-Agnostic	Order-Preserving
Continuity Equation	Eq. 24	Eq. 40	Eq. 46
Within-length $\mathcal{B}$	0	$v\cdot\nabla$ (or discrete CTMC)	$v\cdot\nabla$ (or discrete CTMC)
Outgoing jump Q	$\lambda(k)\delta_{k+1}$	$\lambda(y)\rho(\mathrm{d}m\|y)$	$\sum_{i=0}^{k}\lambda^i(y)\rho^i(\mathrm{d}a\|y)$

The continuity equations in Eq. 24, 40, and 46 describe the evolution of the three GPFlow variants. Table 4 lists the corresponding *outgoing* generators. It is important to distinguish the jump measure from its appearance in the forward equation. At a current state y, the generator contains one outgoing insertion measure:

$$Q_t(\mathrm{d}y'|y) = \lambda_t(y)\int\delta_{I(y,m)}(\mathrm{d}y')\rho_t(\mathrm{d}m|y). \tag{62}$$

Its total mass is $\lambda_t(y)$. Consequently, the adjoint forward equation contains an outgoing term $-\lambda_t(y)p_t(y)$ and an incoming term obtained by integrating $p_t(\bar{y})Q_t(\mathrm{d}y|\bar{y})$ over predecessor states $\bar{y}$. The predecessor inflow is not a second jump measure at y. For length-only GPFlow, Eq. 62 reduces to $Q_t(\mathrm{d}k'|k) = \lambda_t(k)\delta_{k+1}(\mathrm{d}k')$.

This also clarifies the relation to a general generator-matching formulation. A general jump measure may move arbitrary parts of an existing continuous state. In GPFlow, its support is restricted to length-increasing successors $I(y,m)$: the jump adds a new component sampled from ρ_t, while existing components evolve between jumps through the within-length generator $\mathcal{B}_t$. Thus, λ_t and ρ_t are always coupled as the rate and normalized insertion kernel of the same event.

Let $Q_t = \lambda_t\rho_t$ and $\tilde{Q}_t = \tilde{\lambda}_t\tilde{\rho}_t$, suppressing the deterministic insertion map for clarity. Because ρ_t and $\tilde{\rho}_t$ are probability kernels, the generalized KL divergence between the positive jump measures decomposes exactly as:

$$\mathcal{D}_{\mathrm{KL}}(Q_t\|\tilde{Q}_t) = \tilde{\lambda}_t - \lambda_t + \lambda_t\log\frac{\lambda_t}{\tilde{\lambda}_t} + \lambda_t\int\rho_t(\mathrm{d}m)\log\frac{\mathrm{d}\rho_t}{\mathrm{d}\tilde{\rho}_t}(m) \tag{63}$$

$$= \mathcal{D}_{\mathrm{KL}}(\lambda_t\|\tilde{\lambda}_t) + \lambda_t\mathcal{D}_{\mathrm{KL}}(\rho_t\|\tilde{\rho}_t). \tag{64}$$

There is therefore no double counting: the Poisson rate NLL supplies the first term, and the insertion-kernel NLL supplies the second. With exact NLL parameterizations, their sum is the jump-measure term in Theorem 4.1. The mean-regression approximation used for continuous insertion values is a practical surrogate.

A.8 Connection to EditFlow

In this section, we demonstrate the connection between GPFlow, when instantiated only on the discrete modality, and existing work such as EditFlow [23]. We first note that GPFlow can adopt the "deletion" operation by extending the generalized Poisson process to an inhomogeneous birth-death process. For simplicity, we now consider EditFlow with only insertion and substitution. Recall that the EditFlow is a variable-length discrete flow matching model whose loss is defined as:

$$\mathcal{L}_{\text{edit}} = \mathbb{E}_{t,S_t,X_t}\left[\sum_{k=1}^{X_t}u_t(\cdot|S_t) - \sum_{k=1}^{n_{\text{edit}}}\frac{\dot{\kappa}_t}{1-\kappa_t}\log u_t(S_1|S_t)\right], \tag{65}$$

where $n_{\text{edit}} \geq X_1 - X_t$ is the number of edit operations, including insertions ($n_{\text{ins}} = X_1 - X_t$) and substitutions ($n_{\text{sub}} = n_{\text{edit}} - n_{\text{ins}}$). EditFlow predicts the token-wise mixed-type "vector field" $u_\theta(S_t, t)$, which is parameterized as:

$$u_\theta^{\text{ins}}(S_t, t) := \lambda_\theta^{\text{ins}}(S_t, t) Q_\theta^{\text{ins}}(s|S_t, t), \tag{66}$$

$$u_\theta^{\text{sub}}(S_t, t) := \lambda_\theta^{\text{sub}}(S_t, t) Q_\theta^{\text{sub}}(s, s'|S_t, t), \tag{67}$$

where λ_θ is the parameterized insertion/substitution rate, Q_θ^{ins} is the Markov chain transition matrix representation of the probability of inserting a new token s, and Q_θ^{sub} is the probability of substituting the old token s' with a new token s. We can first decompose the loss into the insertion and substitution parts as:

$$\begin{aligned}\mathcal{L}_{\text{edit}} = \mathcal{L}_{\text{ins}} + \mathcal{L}_{\text{sub}} = \; & \mathbb{E}_{t,S_t,X_t}\left[\sum_{k=1}^{X_t} u_t^{\text{ins}} - \sum_{k=1}^{n_{\text{ins}}} \frac{\dot\kappa_t}{1-\kappa_t} \log u_t^{\text{ins}}(s_1)\right] \\ & + \mathbb{E}_{t,S_t,X_t}\left[\sum_{k=1}^{X_t} u_t^{\text{sub}} - \sum_{k=1}^{n_{\text{sub}}} \frac{\dot\kappa_t}{1-\kappa_t} \log u_t^{\text{sub}}(s_1, s')\right].\end{aligned} \tag{68}$$

We first note that for both Q_θ, the distribution property requires $\sum_s Q(s|\cdot) = 1$. In this way, the first term in Eq. 65 reduces to $\sum \lambda_\theta$, the total rate described in the order-preserving GPFlow in Appendix B.1. Using $\log u_\theta = \log \lambda_\theta + \log Q_\theta$, we have:

$$\mathcal{L} = \underbrace{\sum_{k=1}^{X_t} \lambda - \sum_{k=1}^{n} \frac{\dot\kappa_t}{1-\kappa_t} \log \lambda}_{\text{GP}} \underbrace{- \sum_{k=1}^{n} \frac{\dot\kappa_t}{1-\kappa_t} \log Q}_{\text{reconstruction}}. \tag{69}$$

For the insertion loss, it immediately becomes clear that the first two terms together coincide with $\mathcal{L}_{\text{GP}}$ in Eq. 4 as $n_{\text{ins}} = X_1 - X_t$, and the last terms coincide with the cross-entropy reconstruction loss $\mathcal{L}_{\text{rec}}$ in Eq. 91 using the formulation in Gat et al. [19]. For the substitution loss, since substitutions do not alter sequence lengths, the rate component can be effectively coalesced into a single logit component $\hat{Q}_\theta$. Specifically, the survival of a substitution event is equivalent to a substitution with $s = s'$. In this way, we may assume the event always happens while compensating with a non-zero $\hat{Q}_\theta(s, s) := \lambda_\theta^{\text{sub}} > 0$ to allow staying in the same state. Therefore, throwing away the unnecessary rate part, the substitution loss becomes:

$$\mathcal{L}_{\text{sub}} = -\sum_{k=1}^{n} \frac{\dot\kappa_t}{1-\kappa_t} \log \hat{Q}, \tag{70}$$

which is the standard loss for training a discrete flow matching model in Gat et al. [19], serving as $\mathcal{L}_{\text{FM}}$ to refine the discrete modality in our GPFlow formulation. Combining the losses together, we get $\mathcal{L}_{\text{edit}} = \mathcal{L}_{\text{ins}} + \mathcal{L}_{\text{sub}} = \mathcal{L}_{\text{GP}} + \mathcal{L}_{\text{rec}} + \mathcal{L}_{\text{FM}}$, which is but a special case of our GPFlow loss in Eq. 8 on a single discrete modality.

B Algorithmic Details

In this section, we provide additional details on the algorithms, model design, and general techniques we used to improve our GPFlow. Explicit formulas for the losses are also provided.

B.1 Order-Preserving Generalized Poisson Flow

For multimodal GPFlow, a natural setting arises when the ordering of a sequence matters in practice, as in protein design, where amino acids have a natural ordering. In the main context, we treat all Poisson events as an unordered set and model the counting process, which is suitable for order-agnostic generative tasks such as molecule generation. We now discuss how to extend GPFlow to ordered sequential data in a natural way.

For an ordered clean state $Y_1 = (X_1, S_1)$ with $X_1 = L$ and $S_1 = (S_1^1, \ldots, S_1^L)$, consider a current state $Y_t = y = (k, s)$ whose k components correspond to retained target indices $r_1 < \cdots < r_k$. Set

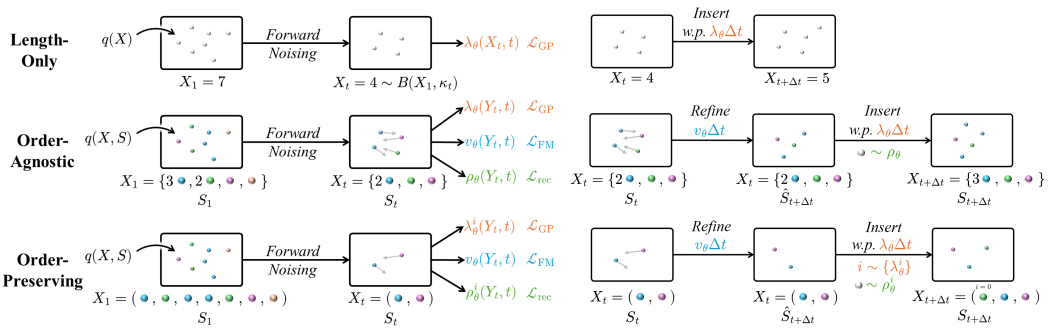


Figure 6: Training procedure (**left**) and one sampling step (**right**) for three variants of GPFlow: **Length-Only** (Section 3.1), **Multimodal Order-Agnostic** (Section 3.2), and **Multimodal Order-Preserving** (Appendix B.1).

$r_0 = 0$ and $r_{k+1} = L + 1$. The $L - k$ unretained target indices are partitioned among the $k + 1$ insertion slots by:

$$\Omega_i := \{\ell \in \{1, \dots, L\} \setminus \{r_1, \dots, r_k\} : r_i < \ell < r_{i+1}\}, \quad 0 \le i \le k. \tag{71}$$

Thus, Ω_i contains exactly the upcoming components that belong between the i-th and $(i + 1)$-st retained components, with $i = 0$ and $i = k$ representing the beginning and end. For each unretained index ℓ, let $a_\ell \in \mathsf{S}$ denote its component value along the conditional path at time t.

Consistent with Appendix A.4, define the insertion-variable space at y as:

$$\mathsf{M}(y) := \{0, \dots, k\} \times \mathsf{S}, \quad m = (i, a), \tag{72}$$

where i is the insertion slot and a is the new component value. For $s = (s^1, \dots, s^k)$, the general insertion map specializes to:

$$I(y, (i, a)) =: I_i(y, a) := \left(k + 1, (s^1, \dots, s^i, a, s^{i+1}, \dots, s^k)\right), \tag{73}$$

with the natural empty-prefix or empty-suffix convention at $i = 0$ or $i = k$.

For the conditional thinning path indexed by Y_1, slot i has rate

$$\lambda_t^i(y|Y_1) = |\Omega_i| h_t, \quad \lambda_t(y|Y_1) = \sum_{i=0}^{k} \lambda_t^i(y|Y_1) = (L - k) h_t, \tag{74}$$

and $\rho_t^i(\mathrm{d}a|y, Y_1)$ is the normalized sampling distribution over $\{a_\ell : \ell \in \Omega_i\}$. When $\lambda_t(y|Y_1) > 0$, these slot-specific quantities define the joint insertion-variable distribution:

$$\rho_t(i, \mathrm{d}a|y, Y_1) := \frac{\lambda_t^i(y|Y_1)}{\lambda_t(y|Y_1)} \rho_t^i(\mathrm{d}a|y, Y_1). \tag{75}$$

Pushing this distribution through Eq. 73 gives the conditional outgoing jump measure:

$$Q_t(A|y, Y_1) = \sum_{i=0}^{k} \lambda_t^i(y|Y_1) \int_{\mathsf{S}} \mathbb{1}\{I(y, (i, a)) \in A\} \rho_t^i(\mathrm{d}a|y, Y_1), \tag{76}$$

which is the order-preserving specialization of Eq. 32. Theorem A.2 posterior-marginalizes these slot-specific rates and distributions.

During learned sampling, the same factorization first simulates an event with total rate $\lambda_\theta(y, t) = \sum_{i=0}^{k} \lambda_\theta^i(y, t)$, then chooses slot i with probability $\lambda_\theta^i(y, t)/\lambda_\theta(y, t)$, samples $a \sim \rho_\theta^i(\cdot|y, t)$, and updates the state through $I(y, (i, a)) = I_i(y, a)$. This process generates the target order-preserving distribution, as guaranteed by Theorem A.2. In this way, the generalized Poisson NLL loss in Eq. 4 is adapted as the summation over $k + 1$ bins:

$$\mathcal{L}_{\mathrm{GP}} = \mathbb{E}_{t, Y_1 \sim q, Y_t \sim p_t(\cdot|Y_1)} \left[\sum_{i=0}^{X_t} \left(\lambda_t^i - h_t |\Omega_i| \log \lambda_t^i \right) \right], \tag{77}$$

and the reconstruction loss of the sampling distributions in Eq. 7 is similarly adapted as:

$$\mathcal{L}_{\text{rec}} = \mathbb{E}_{t,Y_1\sim q,Y_t\sim p_t(\cdot|Y_1)}\left[-h_t \sum_{i=0}^{X_t}\sum_{\ell\in\Omega_i}\log\rho_t^i(a_\ell|Y_t)\right]. \tag{78}$$

We outline the training and sampling procedures for the order-preserving GPFlow in Algorithm 2 and 3, respectively, and adopt it in our experimental setup for protein design. A pictorial comparison of the training and sampling stages of our GPFlow variants is shown in Figure 6 across different scenarios. We also note that the previous discussions for both order-agnostic and order-preserving GPFlow can be easily generalized to an arbitrary number of length-dependent modalities $S = \{S^{(m)}\}_{1\le m\le M} = \{S^{(m),i}\}_{1\le m\le M, 1\le i\le X}$. For simplicity and clarity of notation, we will omit the modality superscripts.

Algorithm 2 Training Order-Preserving GPFlow

while not converged **do**
  Sample $t\sim[0,1]; Y_1 = (X_1, S_1)\sim q(X,S)$.
  Sample $S_t\sim p_t(S)$ for each $S_t^i, 1\le i\le X_1$.
  For each position ℓ, delete S_t^ℓ with probability $1-\kappa_t$ to obtain $\hat{S}_t$ and $X_t = |\hat{S}_t|, Y_t = (X_t, \hat{S}_t)$.
  Predict the rates $\lambda_\theta^i(Y_t,t)$, the length-dependent modality vector field $v_\theta(Y_t,t)$, and the sampling distributions $\rho_\theta^i(Y_t,t), 0\le i\le X_t$.
  Optimize the loss $\mathcal{L}$ in Eq. 8.

Algorithm 3 Sampling from Order-Preserving GPFlow (Euler)

$Y_0 = (X_0 = 0, S_0 = \emptyset)$.
for $n\leftarrow 1,2,\ldots,N$ **do**
  $t\leftarrow t_n, \Delta t\leftarrow t_{n+1}-t_n$, where $t_{N+1} := 1$.
  Predict the rates $\lambda_\theta^i(Y_t,t)$, the vector field $v_\theta(Y_t,t)$, and the sampling distributions $\rho_\theta^i(Y_t,t)$ for $0\le i\le X_t$.
  Advance the generative flow for each S_t with v_θ to obtain $\hat{S}_{t+\Delta t}$.
  Set $\hat{Y}_{t+\Delta t} := (X_t, \hat{S}_{t+\Delta t})$ and $\lambda_\theta := \sum_{i=0}^{X_t}\lambda_\theta^i(Y_t,t)$.
  Sample an insertion event with probability $\lambda_\theta\Delta t$.
  If an event happens, choose slot i with probability $\lambda_\theta^i/\lambda_\theta$, sample $a\sim\rho_\theta^i(\cdot|Y_t,t)$, and set $Y_{t+\Delta t} = I_i(\hat{Y}_{t+\Delta t}, a)$; otherwise set $Y_{t+\Delta t} = \hat{Y}_{t+\Delta t}$.
Return: $Y_1 = (X_1, S_1)$.

B.2 Training and Sampling Pseudocode

We provide additional pseudo-code for training and sampling length-only (Algorithm 5, 6), order-agnostic (Algorithm 4), and conditional order-preserving (Algorithm 7, 8) GPFlow. The three unconditional variants are summarized in Figure 6, which highlights the differences in the training objectives and sampling predictions.

Algorithm 4 Training Multimodal GPFlow

while not converged **do**
  Sample $t\sim[0,1]; Y_1 = (X_1, S_1)\sim q(X,S)$.
  Sample $S_t\sim p_t(S)$ for each $S_t^i, 1\le i\le X_1$.
  For each element $S_t^i\in S_t$, delete it with probability $1-\kappa_t$ to obtain $\hat{S}_t$ and $X_t = |\hat{S}_t|, Y_t = (X_t, \hat{S}_t)$.
  Predict the rate $\lambda_\theta(Y_t,t)$, the length-dependent modality vector field $v_\theta(Y_t,t)$, and the sampling distribution $\rho_\theta(Y_t,t)$.
  Optimize the loss $\mathcal{L}$ in Eq. 8.

Algorithm 5 Training Length-Only GPFlow

while not converged **do**
  Sample $t \sim [0,1]; X_1 \sim q(X)$.
  Sample $X_t \sim B(X_1, \kappa_t)$.
  Predict the rate $\lambda_\theta(X_t, t)$ and optimize the loss $\mathcal{L}_{\text{PP}}$ in Eq. 4.

Algorithm 6 Sampling from Length-Only GPFlow (Euler)

$X_0 = 0$.
for $i \leftarrow 1, 2, \ldots, N$ **do**
  $t \leftarrow t_i, \Delta t \leftarrow t_{i+1} - t_i$, where $t_{N+1} := 1$.
  Predict the rate $\lambda_\theta(X_t, t)$.
  Sample an event with a probability of $\lambda_\theta \Delta t$.
  If an event happens, $X_{t+\Delta t} = X_t + 1$; else $X_{t+\Delta t} = X_t$.
Return: X_1.

Algorithm 7 Training Conditional Order-Preserving GPFlow

while not converged **do**
  Sample $t \sim [0,1]; Y_1 = (X_1, S_1) \sim q(X, S)$.
  Sample condition $Y_0 = (X_c, S_c) \sim q_c(X, S | X_1, S_1)$ with clean values.
  Sample $S_t \sim p_t(S)$ for each non-conditioning position i, $S_t^i, 1 \le i \le X_1$.
  For each non-conditioning position i, delete S_t^i with probability $1 - \kappa_t$ to obtain $\hat{S}_t$ and $X_t = |\hat{S}_t|, Y_t = (X_t, \hat{S}_t)$.
  Predict the rates $\lambda_\theta^i(Y_t, t)$, the length-dependent modality vector field $v_\theta(Y_t, t)$, and the sampling distributions $\rho_\theta^i(Y_t, t), 0 \le i \le X_t$.
  Optimize the loss $\mathcal{L}$ in Eq. 8.

Algorithm 8 Sampling from Conditional Order-Preserving GPFlow (Euler)

Input: $Y_0 = (X_c, S_c)$.
for $i \leftarrow 1, 2, \ldots, N$ **do**
  $t \leftarrow t_i, \Delta t \leftarrow t_{i+1} - t_i$, where $t_{N+1} := 1$.
  Predict the rates $\lambda_\theta^i(Y_t, t)$, the vector field $v_\theta(Y_t, t)$, and the sampling distributions $\rho_\theta^i(Y_t, t)$ for $0 \le i \le X_t$.
  Advance the generative flow for each S_t with v_θ to obtain $\hat{S}_{t+\Delta t}$ for each non-conditioning position i.
  Zero out the rates where insertions are not allowed (e.g., inside a contiguous condition sequence).
  Simulate the Poisson process by sampling an event on $X_{t+\Delta t}$ with a probability of $\lambda_\theta \Delta t$ where $\lambda_\theta = \sum_i \lambda_\theta^i$.
  If an event happens, choose the insertion position i with probabilities proportional to $\{\lambda_i\}$ and insert the new feature value $s_i \sim \rho_\theta^i$ into $\hat{S}_{t+\Delta t}$ to obtain $S_{t+\Delta t}$.
  Update $Y_{t+\Delta t} = (X_{t+\Delta t}, S_{t+\Delta t})$.
Return: $Y_1 = (X_1, S_1)$.

In all sampling algorithms, the sequence of discretization time steps $0 \le t_1 < t_2 < \cdots < t_N < t_{N+1} = 1$ is defined by an inference-time scheduler, and more advanced solvers (e.g., the second-order Heun solver) can be readily applied.

B.3 Flow Parameterization

Continuous Euclidean Modality Existing works have explored fixed-length diffusion or flow-based generative models for both continuous and discrete modalities, whose loss can be adapted as

our $\mathcal{L}_{\text{FM}}$ for refining the existing length-dependent elements. For example, consider a flow matching model with the optimal-transport path [39] $s_t = (1-t)s_0 + ts_1$, where $s_0 \sim \mathcal{N}(0, \sigma_0^2)$. The flow matching loss is

$$\mathcal{L}_{\text{FM}} = \mathbb{E}_{t,s_0\sim p_0,s_1\sim q}[\|v_\theta(s_t,t) - (s_1 - s_0)\|^2], \tag{79}$$

where $v_\theta(s_t, t)$ is the vector field predictor. We follow Proteina in using this loss to refine the continuous CA coordinates in our structure design model. For sampling from such a continuous vector field, a simple Euler update reads:

$$s \leftarrow s + v_\theta \Delta t. \tag{80}$$

We may also employ more complex sampling algorithms that incorporate higher-order information or stochasticity (e.g., SDE sampling). For example, one sampling step in Proteina [21] reads:

$$s \leftarrow s + (v_\theta + g_t s_\theta)\Delta t + \sqrt{2g_t\gamma\Delta t}\,\varepsilon. \tag{81}$$

where $s_\theta(s_t, t) := (tv_\theta(s_t, t) - s_t)/(1-t)$ is the score function, g_t is the score scheduler, $0 < \gamma < 1$ is the noise scale, and $\varepsilon \sim \mathcal{N}(0, 1)$. We also adopt this SDE sampling for the coordinates.

Continuous Riemannian Modality For Riemannian modalities such as rotations (which lie in the 3D rotation group $\mathrm{SO}(3)$) and torsion angles (which lie in the flat torus $\mathbb{T}$) in the peptide co-design task, the standard Riemannian flow matching loss [12] is applied:

$$\mathcal{L}_{\text{FM}} = \mathbb{E}_{t,s_0\sim p_0,s_1\sim q}[\|v_\theta(s_t,t) - u_t(s_t|s_1)\|_g^2], \tag{82}$$

where geodesic interpolation is used as the noising process $s_t := \exp_{s_0}(\kappa_t \log_{s_0} s_1)$ with a differentiable scheduler κ_t. The $\exp$ denotes the exponential map, $\log$ the logarithm map, and $\|\cdot\|_g$ the Riemannian norm. The conditional vector field u_t can be calculated in closed form as

$$u_t(s_t|s_1) = \frac{\dot\kappa_t}{1-\kappa_t} \log_{s_t} s_1. \tag{83}$$

Sampling from Riemannian flow matching follows the Euler discretization as:

$$s \leftarrow \exp_s(v_\theta \Delta t). \tag{84}$$

The mathematical expression for the Riemannian operations on $\mathrm{SO}(3)$ and $\mathbb{T}$ is known in closed form. We refer interested readers to, e.g., Appendix A of Li et al. [34] for full details.

Discrete Modality For continuous-state discrete flow matching models [14] that embed the discrete token into a continuous modality, the previous discussion of continuous-modality losses applies directly [27, 13, 18]. For example, the simplex-based GPFlow variant for peptide design adopts the soft logits $s_1 = K(2\delta_{c_1} - 1)$, where c_1 is the corresponding amino acid type and $K = 5$. s_0 is sampled from $\mathcal{N}(0, KI)$, and Euclidean geometry is adopted. Following Li et al. [34], the cross-entropy loss is used instead of the vector field matching loss:

$$\mathcal{L}_{\text{FM}} = \mathbb{E}_{t,s_0\sim p_0,s_1\sim q}[\mathrm{CE}(f_\theta(s_t,t), c_1)], \tag{85}$$

where $f_\theta(s_t, t)$ is the logit predictor (probability denoiser).

For the discrete modality that relies on a continuous-time Markov chain, the intermediate noisy s_t is a discrete token rather than continuous logits. The discrete flow matching loss [19, 50] reads:

$$\mathcal{L}_{\text{FM}} = \mathbb{E}_{t,s_0\sim p_0,s_1\sim q}\left[\sum_{s_t\neq s_1} \mathrm{CE}(f_\theta(s_t,t), s_1)\right], \tag{86}$$

where the noisy s_t is sampled from some pre-defined probability path on the noisy logits x_t:

$$s_t := \begin{cases} \sim \mathrm{Cat}(x_t), & \text{w.p. } 1-t, \\ s_1, & \text{w.p. } t. \end{cases} \tag{87}$$

A simple yet effective practice follows mask diffusion [50]: each noisy token starts with a special mask token m with $s_0 \equiv \delta_m$, and the FM loss is computed over all mask tokens. During sampling, a single mask diffusion update reads:

$$s \leftarrow \begin{cases} \sim \mathrm{Cat}(f_\theta(s_t,t)), & \text{if } s_t = m \text{ and w.p. } \Delta t/(1-t), \\ s_t, & \text{else.} \end{cases} \tag{88}$$

Notably, we may consider a special case in which the insertion element type is fixed to be the mask token, leaving no reconstruction loss $\mathcal{L}_{\text{rec}} = 0$ for the discrete modality. In this case, the model completely relies on the non-zero mask diffusion loss $\mathcal{L}_{\text{FM}}$ to determine the final discrete type.

B.4 Sampling Distribution Parameterization

The marginal sampling distribution in Theorem 3.3 and A.2 is a mixture distribution of the forward noisy length-dependent modality from which sampling is feasible (when conditioned on the clean target). Therefore, we can rely on another generative model for modeling such a distribution, whose general NLL loss form is described in Eq. 7 and 78.

Continuous Euclidean Modality We first consider the continuous modality case where directly regressing on the sampling marginal ρ_t is often intractable. We may rely on another flow matching head or any generative model to approximate ρ_t parametrically, using the current state S_t and the noisy element to be inserted as inputs. While such a nested flow approach is theoretically preferred, it requires simulating an additional full flow sampling for each Poisson event, thereby substantially slowing the sampling process. In practice, we found that using the MSE loss to regress the mean behavior of S_t already performs well enough with the existing later denoising steps. The simplified loss can be written as:

$$\mathcal{L}_{\text{rec}} = \mathbb{E}_{Y_1, Y_t}\left[h_t \sum_{k=1}^{X_1 - X_t} (f_\theta(Y_t) - S_t^k)^2\right]. \tag{89}$$

Continuous Riemannian Modality The simplified reconstruction loss can be easily extended to Riemannian generative modeling (e.g., rotations and torsion angles in peptide design), which takes:

$$\mathcal{L}_{\text{rec}} = \mathbb{E}_{Y_1, Y_t}\left[h_t \sum_{k=1}^{X_1 - X_t} d_g^2(f_\theta(Y_t), S_t^k)\right], \tag{90}$$

where $d_g^2(x, y) = \| \log_x y \|_g^2$ is the squared geodesic distance.

Discrete Modality For Markov-chain-based discrete flow matching models, Gat et al. [19] has established a connection between the discrete vector field and the probability flow over categorical data, where the sampling marginal can be obtained in closed form. Following this, the KL-divergence (cross-entropy loss) is used to train the discrete flow, and the reconstruction loss can be calculated as a summation of weighted cross-entropy losses as:

$$\mathcal{L}_{\text{rec}} = \mathbb{E}_{Y_1, Y_t}\left[h_t \sum_{k=1}^{X_1 - X_t} \text{CE}(f_\theta(Y_t), S_t^k)\right], \tag{91}$$

where f_θ is the logit predictor. Thus, unlike the continuous case, the discrete case is always simulation-free, as the learned logits already encode the marginalized discrete sampling distribution.

In contrast to the flow-oriented loss discussed in the previous subsection, we may consider another special case in which the element type is determined upon insertion, yielding no refinement loss with $\mathcal{L}_{\text{FM}} = 0$. In this approach, the sampling distribution serves a dual role in determining the discrete types without further refinement. This parameterization and sampling scheme are adopted for our sequence design model. Nonetheless, one may also allow token refinement relying on the discrete diffusion [2, 50] or discrete flow matching [19] loss $\mathcal{L}_{\text{FM}}$ to further refine the discrete modality.

For all time-dependent modalities, the training noise schedulers and inference time sampling schedulers for the flow and reconstruction parts can be either coupled with the rate scheduler or separate. We typically use different schedulers for the rate and length-dependent modality to allow greater flexibility during sampling. In this way, time steps across modalities are not shared during training, and separate time embeddings are fed into the model.

B.5 τ-Leaping

One practical limitation of existing rate-based variable-length generative models is their slower sampling speed compared to standard fixed-length flow sampling, because we must simulate the inhomogeneous Poisson process at each time step, which is more sensitive. For example, EditFlow [23] required 5,000-10,000 sampling steps to achieve reasonable performance, which is significantly larger than the average sequence length and the typical number of steps used in fixed-length diffusion or flow matching models. To speed up the sampling process, we consider simulating multiple Poisson

events within a small time step τ by leveraging the τ-*leaping* method [22]. Specifically, instead of sampling one event with a success probability of $\lambda\tau$, the increment is now sampled from a Poisson distribution with a rate of $\lambda\tau$:

$$X_s - X_t \sim \mathcal{P}\left(\lambda(X_t, t)\tau\right), \quad \tau = s - t. \tag{92}$$

Note that we have $\Pr(X_s - X_t = 0) = \exp\left(-\lambda\tau\right)$, which is close to $1 - \lambda\tau$ when $\lambda\tau$ is small, recovering the standard simulation procedure. Nonetheless, when $\lambda\tau$ is large, the τ-leaping method can significantly reduce the number of sampling steps while still maintaining a good approximation to the original process. As an example, the ground-truth marginal rate on the PDB dataset at $t = 0$ is the average sequence length, which is approximately 166. With a standard 400-step sampling setup in Proteina, the initial rate is approximately 0.4, at which point the approximation will fail.

In addition, we also allow for a more general form of τ-leaping, where simultaneous insertions can happen at multiple places independently according to their own rates and sampling distributions [36]. Such a multi-place insertion better approximates the original process than the Bernoulli approximation. Nonetheless, we note that τ-leaping relies on the assumption that a single event does not significantly affect predictions, which does not necessarily hold even for discrete diffusion with a fixed time step [49]. In our scenario, where length is also variable, such an approximation error may be more pronounced. Despite this, we empirically find that both forms of τ-leaping improve performance and efficiency in the 400-step generation setup.

B.6 Additional Related Work

Here, we provide additional related work on variable-length generative frameworks that primarily focus on text or image generation, as well as other autoregressive frameworks that naturally support variable-length protein generation. Although these works don't directly adapt to general protein structure design or co-design, they still offer useful methodological insights.

Protein Design Models. Beyond the structure generative models discussed in the main text, related approaches address protein sequence design, multimodal protein generation, and peptide design. ProteinMPNN [16] and ESM-IF1 [28] perform inverse folding by generating amino acid sequences conditioned on backbone structures. ESM3 [24] jointly models protein sequences, structures, and functional annotations through a token-based framework, enabling protein generation with multimodal conditioning. Generative modeling has also been applied to the design of peptides targeting specific proteins. PepMimic [32] designs peptide binders by mimicking the binding interfaces of existing protein binders. PepHAR [35] introduces a hotspot-driven autoregressive framework for peptide design through multi-fragment extension and iterative refinement. PocketXMol [45] provides a unified atom-level generative framework based on atomic interactions, supporting diverse molecular design tasks, including peptide generation.

Variable-length Generative Frameworks. We further discuss alternative tokenization and generation schemes for proteins, together with variable-length generative frameworks developed in other domains. PAR [47] adopts a multi-scale framework that generates protein backbones through coarse-to-fine next-scale prediction, combining an autoregressive transformer with a flow-based decoder. Although it compresses protein structures into fixed-length coarse token representations, generation still requires a specified protein length. APT [17] uses an adaptive tokenizer in which successive tokens encode increasingly fine-grained details of a global structural representation. The final protein length is inferred from the first few coarse-grained global tokens during generation. Autoregressive protein language models [56, 46] generate amino acid sequences token by token, naturally supporting variable-length sequence generation. Autoregressive modeling has also been extended to structure tokens to incorporate structural information into protein generation [20]. In other domains, OneFlow [43] combines insertion-based Edit Flows for discrete text tokens with flow matching for continuous image latents, enabling variable-length, interleaved text–image generation.

C Experiment Details

In this section, we detail the datasets, task-specific architectures, and evaluation pipelines. The two structure-based generation tasks are built on the Proteina base model (with different feature variants) and use the same datasets; the two sequence-design tasks share a separate backbone family, and

peptide design is described in its own subsection. We therefore organize this section by dataset/model architecture.

Proteins are naturally ordered according to their amino acid sequence. We always use the order-preserving GPFlow for generative modeling in all tasks. Table 5 summarizes the per-task modeling choices, including the modality geometry, the flow and reconstruction losses, the insertion scheduler, and the base model architecture.

Table 5: Per-task modeling choices for GPFlow. All tasks use the order-preserving variant.

Task	Length-dependent modality	$\mathcal{L}_{\text{FM}}$	$\mathcal{L}_{\text{rec}}$	Scheduler κ_t	Backbone
Unconditional structure	CA coordinates ($\mathbb{R}^3$)	OT-FM (Eq. 79)	mean MSE (Eq. 89)	$\min(1, t/0.6)$	Proteina
Structure motif scaffolding	CA coordinates ($\mathbb{R}^3$)	OT-FM (Eq. 79)	mean MSE (Eq. 89)	$\min(1, t/0.6)$	Proteina
Unconditional sequence	residue type (discrete)	none ($\mathcal{L}_{\text{FM}}$=0)	CE (Eq. 91)	t^3 (localized)	DiT
Sequence motif scaffolding	residue type (discrete)	none ($\mathcal{L}_{\text{FM}}$=0)	CE (Eq. 91)	t^3 (localized)	DiT
Peptide co-design	$\mathrm{SO}(3)$, $\mathbb{R}^3$, $\mathbb{T}$, type (simplex)	RFM & CE	geodesic mean MSE	linear	PepFlow

C.1 Protein Structure Generation

C.1.1 Data Specification

The datasets we use to train the model include PDB [5], the experimentally validated protein structure database, and AFDB [29], the AlphaFold2-predicted structure database that contains more structures. For the PDB dataset, we use the Proteina preprocessing script to filter entries that have a single chain, contain no non-canonical amino acids, and have lengths between 50 and 256. The data are then partitioned using a 50% sequence-similarity threshold to prevent information leakage within the same cluster, yielding 6822 distinct clusters, for a total of 27k training structures. During training, we follow Proteina's approach to uniformly sample one cluster and then a member of that cluster.

For AFDB, we follow Proteina to filter and cluster the original 21M AFDB structures using sequence-based MMseqs2 [52] and structure-based FoldSeek [55], resulting in 713k structures. Unlike PDB, we use the standard uniform sampling strategy. As AFDB contains much longer proteins, each structure is dynamically cropped to a maximum sequence length of 512 to avoid memory issues during training.

C.1.2 Model Specification

Architecture. We build each structure model on top of its corresponding 60M Proteina [21] architecture and add heads to the final hidden representation to learn per-position rates and sampling distributions, thereby increasing the total number of trainable parameters to 65 M. In both unconditional generation and motif scaffolding, the only length-dependent variable is the continuous CA coordinates.

For motif scaffolding, we adapt Proteina's dedicated motif architecture variant, which includes additional information on the motif mask, coordinates, and pairwise distances. This architecture is not checkpoint-compatible with the unconditional Proteina model, so neither method transfers an unconditional checkpoint into the motif experiment. All residue indices are renumbered to avoid exposing their original positions. We additionally inject sequence embeddings for the motif residues (if available), with other non-motif residues set to unknown (X). We adopt a hard constraint in an inpainting fashion: the motif coordinates and types are never corrupted or modified and incur no loss during either training or sampling.

Training. During training, we generally follow the original setup and hyperparameters in Proteina, where the time step is sampled from $t \sim 0.98\mathrm{Beta}(1.9, 1.0) + 0.02U[0, 1]$. For the length modality, we adopt the early-completion principle used by TDDM [10] and set $\kappa_t := \min\{1, t/0.6\}$. The length time $t \sim U[0, 1]$ is sampled independently from the continuous time and is fed to the model as an additional input. As defined in Theorem 3.1, the process is absorbed once $\kappa_t = 1$: for $t \geq 0.6$, all target elements are present, the insertion rate and insertion-loss contribution are set to zero, and only the within-length refinement process remains active. The OT-path-based flow matching loss in Eq. 79, the NLL loss for the Poisson process in Eq. 4, and the mean reconstruction loss in Eq. 89 are used with weights $w_{\text{FM}} = w_{\text{rec}} = 1$. For motif scaffolding, we use the conditional generation

setup, in which the motif CA coordinates remain unchanged and are never deleted during the forward noising process. The other losses are identical.

For unconditional generation, two separate 65M models are trained: one with a total batch size of 128 and 300k iterations on PDB, and another with a total batch size of 32 and 1.6M iterations on AFDB. For motif scaffolding, both the 65M GPFlow model and the corresponding Proteina baseline use the AFDB dataset and the same motif augmentation. Our model is trained for 1M iterations with a total batch size of 192. We follow Lin et al. [37] to randomly sample 0-4 contiguous parts (contigs) as motifs during training. All models are trained with the Adam optimizer with a learning rate of 10^{-4}. An exponential moving average (EMA) of the model parameters is maintained with a decay of 0.999 and used for all evaluations. All models were trained on 8 NVIDIA A100 GPUs, taking approximately 2 days on PDB and 5 days on AFDB.

Sampling. During sampling, 400 sampling steps are used following the Proteina setup. For the continuous modality, we also follow Proteina to utilize SDE sampling described by Eq. 81 with $g_t = 1/t$, $\gamma = 0.35$, and an exponential inference scheduler. For the length modality, we use $\kappa_t := \min\{1, t/0.3\}$ with a constant rate scaling factor of 0.7. Consequently, insertions are completed by $t = 0.3$, and the remaining trajectory refines the fixed set of generated coordinates.

For motif scaffolding, motif contigs are provided as the model's initial input. Insertions are allowed between motif contigs and at the beginning and the end, but not within the contiguous motif. The motif parts are centered and remain unchanged during the refinement for the continuous coordinates. We track the lengths of each contig (motifs and scaffolds) to determine the new motif positions for the next input in the iterative sampling process. The same sampling hyperparameters as in the unconditional setup are used, except for a rate scaling factor of 0.6 to compensate for the longer length distribution in AFDB.

We use the τ-leaping technique described in Appendix B.5 to enable high-quality sampling with the 400-step sampling setup. Simultaneous insertion at multiple places is allowed, and multiple insertions at the same place are also allowed, with the count sampled from the corresponding Poisson distribution. In contrast to Proteina, classifier-free guidance (CFG) and self-conditioning are not used.

C.1.3 Evaluation Pipeline

In our evaluations of structure quality and diversity, the following metrics are used:

Designability. For each protein structure, we first use ProteinMPNN [16] to design 8 candidate sequences, which are then folded by ESMFold [38] to give the refolded structures. The self-consistency root-mean-square deviation (scRMSD), defined as the RMSD between the generated and refolded structures, is used as the primary indicator of structural fidelity. A *designable* protein structure in unconditional protein generation is defined as having a minimum scRMSD < 2Å among the 8 refolded candidates. The scRMSD is always calculated on the CA coordinates due to Proteina's CA-only architecture. *Designability* is defined as the proportion of designable protein structures in all generations. The designability score assesses the quality of structural generation, since natural protein structures and scaffolds should be designable.

Scaffold Designability. For motif scaffolding, the following additional metrics are calculated:

- **motifRMSD**, which is the RMSD between the structure of the refolded motif region and the ground-truth motif structure specified as the condition. We extract the motif residues in each refolded structure to calculate motifRMSD. motifRMSD is calculated over the four backbone atoms (N, C, CA, O) instead of CA only.
- **pLDDT** (predicted local distance difference test score) [42], which estimates structural fidelity on a 0-100 scale given the residue sequence. pLDDT is calculated for the ProteinMPNN-redesigned sequences.
- **pAE** (predicted aligned error), which measures the alignment error of residue pairs between the generated and refolded structures.

Additionally, the amino acid sequences in the motif template are fixed and will not be redesigned by ProteinMPNN unless specified as unknown (X) in the template. This ensures that motif information, such as enzymatic residues, is preserved during sequence redesign. Following Watson et al. [58],

a scaffold is considered *successful* if at least one refolded structure (out of 8 candidates) satisfies scRMSD $\leq$ 2Å, motifRMSD $\leq$ 1Å, pLDDT $\geq$ 70, and pAE $\leq$ 5Å. Successful designs are clustered as described below, and each resulting cluster counts as one *unique success*.

Diversity. *Diversity* is defined as the proportion of designable clusters with respect to the total number of designable samples. All predicted structures are clustered with FoldSeek [55] at a specified TM-score threshold [61], TM-score is a length-independent measure of global fold similarity. We calculate the diversity using the following FoldSeek [55] command:

```
foldseek easy-cluster <path_samples> <path_tmp>/res <path_tmp> \
  --alignment-type 1 --cov-mode 0 --min-seq-id 0 \
  --tmscore-threshold <threshold>
```

where the threshold is 0.5 for unconditional generation and 0.6 for motif scaffolding.

Novelty. We assess the structural novelty score via exhaustive search against a structure database with FoldSeek. *Novelty* is defined as the maximum TM-score across all hits for each generated structure, averaged across all generations. We calculate the novelty score using the following FoldSeek command:

```
foldseek easy-search <path_samples> <path_db> out.tsv <path_tmp> \
  --alignment-type 1 --exhaustive-search --tmscore-threshold 0.0 \
  --max-seqs 10000000000 --format-output query,target,alntmscore,lddt
```

where `path_db` is the path prefix to either the full PDB structure database or the filtered 713k AFDB subset, resulting in the two score variants in Table 1. The output TSV file contains results from the exhaustive search against the designated database, which is then processed to extract the highest TM-score.

Secondary structure composition. Per-residue three-state secondary structure is assigned with DSSP [30] via the Biotite library [33]. We report the average per-protein helix (α) and strand (β) fractions in the generated structures, and the coil fraction is given by $1 - \alpha - \beta$.

For unconditional protein design, we calculate all metrics over 500 generations per model. For GPFlow, the 512 lengths are determined by the learned rate. Following the published fixed-length benchmark, each baseline receives 100 requests at each length in $\{50, 100, 150, 200, 250\}$. We note that sampling baseline lengths from the empirical PDB marginal would instead constitute an oracle-conditioned diagnostic because the fixed-length models do not learn $p(L)$. We therefore use the standard grid for the primary end-to-end comparison and report GPFlow's binned designability separately in Appendix E.

For motif scaffolding, we compute the metrics on 1000 generated structures for each motif target, using FoldSeek's clustering utility to identify the number of unique successes described above. For motif tasks 5TRV, 6E6R, 6EXZ, and 7MRX, the original RFDiffusion benchmark provides contig templates across 3 different length ranges. Each contig template specifies the minimum and maximum lengths of each linker segment and of the entire backbone. For fixed-length generative models like Proteina, given a contig template, the model first randomly samples the lengths of each linker segment to satisfy the contig constraints, then fixes the positions of these linker residues and redesigns their structures. In this way, the length distribution in Figure 9 exhibits 3 modes corresponding to the long, medium, and short length ranges. For the baselines, we use the results from the best contig configuration on tasks 5TRV, 6E6R, 6EXZ, and 7MRX. In contrast, GPFlow does not rely on predefined contig templates or length-range constraints. Instead, we sample 1000 backbones conditioned solely on the motif segments, yielding a continuous length distribution without the 3 distinct modes.

C.1.4 Baseline Information

For protein structure design, we compare GPFlow against the following baselines:

Proteina [21] is a CA-only structure design model based on flow matching. For unconditional generation on PDB, a 60M model is retrained on the same PDB dataset using the same training hyperparameters as our GPFlow for 300k iterations, with a total batch size of 128. For AFDB, the official 60M unconditional Proteina checkpoint is used. For motif scaffolding, we use the separate

60M Proteina motif architecture with the same AFDB training set and motif augmentation as our motif GPFlow. Because the unconditional and motif architectures expose different inputs, their checkpoints are not interchangeable; each experiment therefore uses its corresponding Proteina base model. For all Proteina models, we use exactly the same sampling hyperparameters as GPFlow: 400 sampling steps, a noise scale of 0.35, and an exponential scheduler. No self-conditioning is used, as suggested in the default sampling config.

FrameDiff [60] is a Riemannian diffusion model that learns a rotation (orientation) and a translation (position) for each amino acid in a protein. **FrameFlow** [59] further extends it by incorporating Riemannian flow matching, with improved training stability and performance reported in the original work. The results are directly taken from the Proteina paper, which runs inference using the provided checkpoint following the recommended hyperparameters.

RFDiffusion [58] is a diffusion-based protein backbone generative model that leverages the structure prediction power of RoseTTAFold [3]. RFDiffusion also supports motif scaffolding by incorporating motifs as conditions. The unconditional generation results for RFDiffusion are taken directly from the Proteina paper, whereas the motif scaffolding results are taken directly from the original paper.

Genie2 [37] is a diffusion-based generative model with a specialized multi-motif framework that designs co-occurring motifs. The motif scaffolding results are taken directly from the Proteina paper, which follows the recommended sampling hyperparameters in the original paper.

C.2 Protein Sequence Generation

C.2.1 Data Specification

We train our sequence-generation GPFlow on the UniRef50 database [53], containing 41,887,589 representative protein sequences obtained by clustering UniProt [54] sequences at 50% sequence identity. We follow the data processing in EvoDiff [1], which results in a dataset split into 41,546,293 training, 82,929 validation, and 48,941 test sequences. We further filter training sequences to retain only those with lengths between 10 and 1024 residues, yielding 40,471,430 training sequences with a mean length of 247.7 residues. Each sequence is tokenized at the single-residue level over a 21-character alphabet comprising the 20 standard amino acids plus an unknown token X (`ACDEFGHIKLMNPQRSTVWYX`). No subsampling of long sequences is applied; all sequences within the length range are used as-is. For sequence-based motif scaffolding, we finetune on a subset of UniRef50 filtered to sequences with AlphaFold2 [29] pLDDT ≥ 95 and lengths between 10 and 200, yielding 480,143 sequences.

C.2.2 Model Specification

Architecture. We use the Diffusion Transformer (DiT) [44] architecture for both the insertion and deletion models. Each model consists of 14 transformer blocks with a hidden dimension of 1600, 25 attention heads, totaling 642M parameters. Dropout is set to 0.1. Length conditioning is performed via a continuous sinusoidal length embedding over the range $[10, 1024]$.

Training. Both models are trained with AdamW [31] ($\beta_1 = 0.9$, $\beta_2 = 0.95$, no weight decay) and a learning rate of 3×10^{-4} with 2,000 warmup steps followed by a cosine annealing schedule ($\eta_{\min} = 10^{-5}$). Gradient clipping is applied at a norm of 5.0. Dynamic batching is used with a budget of 80,000 tokens per batch with approximate random-length sampling. We use gradient accumulation over 4 micro-batches, yielding an effective batch size of 320,000 tokens. An EMA of the model parameters is maintained with a decay of 0.9999 and is used for all evaluations.

To enable classifier-free guidance during sampling, the length condition c is randomly dropped with probability 0.2 during training, so that a single model learns both the conditional edit rate $u_t^\theta(x|x_t, c)$ and the unconditional rate $u_t^\theta(x|x_t)$. At sampling time, these are combined as described in Eq. 93 and 94. Training was carried out on four NVIDIA RTX PRO 6000 GPUs for 100k gradient updates, taking approximately three days.

Sampling. Generation proceeds by simulating the learned CTMC from the empty sequence ($t = 0$) to the data distribution ($t = 1$) over a uniform time grid with step size $\tau = 1/T$ (Euler solver). Each step independently samples the number of insertions at each position i from a Poisson distribution

with intensity $\lambda^{\text{ins}}_{t,i}(X_t)\tau$, and draws inserted token identities from $Q^{\text{ins}}_{t,i}(\cdot|X_t)$. This first-order approximation corresponds to the linearization $1 - e^{-\lambda\tau} \approx \lambda\tau$.

To mitigate discretization errors that accumulate over these Euler steps, we employ the corrector procedure of Havasi et al. [23], which leverages both the forward rate u_t (insertion) and reverse rate $\tilde{u}_t$ (deletion). The combined rate $(1+\alpha)u_t + \alpha\tilde{u}_t$ preserves the marginal distribution p_t for any corrector strength $\alpha \geq 0$. Note that when $\alpha = 0$, this reduces to the standard forward-only Euler sampler. This allows the introduction of additional self-correction without altering the target distribution. Concretely, at each time step t_k with step size h:

1. **Insertion sub-step.** For each position i of the current sequence X_{t_k}, we sample insertions from a Poisson distribution with intensity $(1+\alpha)\lambda^{\text{ins}}_{t_k,i}(X_{t_k})\tau$. Inserted token identities are drawn from $Q^{\text{ins}}_{t_k,i}(\cdot|X_{t_k})$. This produces an intermediate state Y that overshoots by $\alpha\tau$ in time.
2. **Deletion sub-step.** For each position i of Y, we sample whether to delete that position with probability $\alpha\tilde{\lambda}^{\text{del}}_{t'_k,i}(Y)\tau$, where $t'_k = t_k + (1+\alpha)\tau$ and $\tilde{\lambda}^{\text{del}}$ is the learned reverse (deletion) rate. Deleted positions are removed, yielding $X_{t_{k+1}}$. This pulls back by $\alpha\tau$, so the net time advancement is τ.

We further apply classifier-free guidance (CFG) independently to both the rate magnitude λ and the token distribution Q for the insertion model and only to the latter for the deletion model. Following the formulation in Havasi et al. [23], the guided quantities are

$$\tilde{\lambda}_{t,i}(x_t, c) = \lambda_{t,i}(x_t|c)^{1+w}\, \lambda_{t,i}(x_t)^{-w}, \tag{93}$$

$$\tilde{Q}_{t,i}(a|x_t, c) \propto Q_{t,i}(a|x_t, c)^{1+w}\, Q_{t,i}(a|x_t)^{-w}, \tag{94}$$

where $w \geq 0$ is the guidance weight, c is the target length, and the conditional and unconditional outputs are produced by the same model (with and without the length condition, respectively). Eq. 93 corresponds to log-linear extrapolation in rate space, $\log\tilde{\lambda} = (1+w)\log\lambda(\cdot|c) - w\log\lambda(\cdot)$, amplifying the conditional rate relative to the unconditional baseline. Eq. 94 applies log-linear extrapolation in token-distribution space, analogous to the rate guidance. Together, guidance modulates both *where* edits occur (via $\tilde{\lambda}$) and *what* tokens are inserted (via $\tilde{Q}$). Figure 8 demonstrates the effectiveness of this length control mechanism.

For the results in the main text, we generate 500 sequences using $T = 5{,}000$ steps with corrector strength $\alpha = 1$ and guidance weight $w = 0.75$ for both the forward and reverse models. For the length distribution evaluation (Figure 2(e)), we generate a larger set of 1,000 sequences using $T = 10{,}000$ steps with the forward-only Euler sampler ($\alpha = 0$), nucleus sampling with $p = 0.95$, and temperature annealing from $\tau = 2.0$ to $\tau = 0.1$. For sequence-based motif scaffolding, generation starts from the motif tokens and proceeds for 5,000 steps with corrector strength $\alpha = 1$, with no classifier-free guidance or length conditioning applied.

C.2.3 Evaluation Pipeline

We evaluate GPFlow samples against ground-truth sequences in UniRef50 and samples generated by the baseline models. For all baselines, we follow the DPLM [57] evaluation protocol to generate 500 sequences in total: 50 sequences at each of the 10 fixed lengths $L \in \{100, 200, \ldots, 1000\}$. UniRef50 reference sequences are obtained from 500 randomly sampled training sequences. We fold generated sequences into 3D structures with ESMFold [38]. The predicted structures are then assessed with the metrics detailed below, following a similar protocol in structure design.

Foldability is measured by the sequence pLDDT score, as folded by ESMFold. We report the mean CA pLDDT across all residues of each structure, stratified by sequence length in bins of width 100 residues. **Novelty** is evaluated by performing an exhaustive structural search across the entire PDB database with FoldSeek. The maximum TM-score across all hits defines the pdb-TM, and structures with pdb-TM < 0.5 are classified as structurally novel. **Structural diversity** is reported as the ratio of distinct clusters to the total number of generated structures clustered by FoldSeek with a TM-score threshold of 0.5. **Secondary structure compositions** using DSSP are also reported to provide population-level structural similarity.

C.2.4 Baseline Information

We compare GPFlow against four protein sequence generative models. For each baseline, we use the authors' official model checkpoints and follow the default sampling procedures.

EvoDiff [1] provides two discrete diffusion variants. EvoDiff-OADM uses a masking corruption process that replaces one residue with a mask token at each step until the sequence is fully masked. EvoDiff-D3PM uses a categorical transition-matrix corruption process that introduces substitution noise, so that the terminal distribution approaches a uniform amino-acid sequence. Sampling starts from a uniform amino-acid initialization and iteratively denoises. We report EvoDiff-OADM, the better-performing variant in the original evaluation. The model is trained on UniRef50 sequences 10-1024 residues long. Sequences longer than 1024 residues are subsampled into 1024-residue chunks. In our evaluation, we use the EvoDiff-OADM-640M checkpoint.

DPLM [57] is a discrete diffusion language model instantiated using an absorbing-mask corruption process. The corruption schedule replaces residues with a mask token and treats the mask as an absorbing noise state. In our evaluation, we use the DPLM-650M checkpoint trained on the same dataset as Alamdari et al. [1]. For sampling, we perform a 500-step discrete diffusion denoising process starting from a fully masked sequence, using temperature annealing from 2.0 to 0.1, with stochastic unmasking and top-$p = 0.95$ filtering.

ESM3 [24] is a generative masked language model trained over multiple discrete token tracks that represent sequence, structure, and function. In our experiments, we evaluate only the sequence track, which is trained on a mixture of large-scale protein sequence databases including UniRef, MGnify, JGI, and OAS, together with sequences paired with structure datasets such as PDB, AlphaFoldDB, and ESMAtlas, and further augmented with inverse-folded sequences [24]. We generate sequences using 20-step iterative masked language model decoding with a cosine unmasking schedule and temperature annealing.

SCISOR [4] is a discrete diffusion model specialized for deletions. Its forward noising process inserts random residues into natural protein sequences, and the learned reverse process removes residues to recover natural-like sequences. SCISOR is trained on the same dataset as Alamdari et al. [1]. For unconditional generation, we draw an excess length S from its forward insertion-noise schedule and independently sample L+S residues according to the amino acid frequencies in the training data. We then apply 10-step reverse deletion diffusion at temperature 1.0 to obtain the target length L.

C.2.5 Localized Probability Paths

In Algorithm 2, step 4, each target position i is independently included in z_t with probability κ_t. This corresponds to the standard factorized token-wise mixture path [19], in which the inclusion of each position is an independent Bernoulli trial. While simple, this factorized construction produces partial sequences $\hat{S}_t$ consisting of non-neighboring tokens interspersed with gaps. For long sequences, the local context around each insertion point consists largely of gap positions, limiting the signal for predicting token identities. Adapting the construction introduced in Havasi et al. [23], we replace the factorized probability path with a localized construction that introduces spatial correlation: when a target position becomes present, its neighbors are encouraged to also become present, yielding local neighborhoods in $\hat{S}_t$.

Construction. Let $X_1 = y$ denote the target sequence length. We introduce an auxiliary boolean indicator $\mathbf{m}_t \in \{\texttt{true}, \texttt{false}\}^y$, where $\mathbf{m}_t^j = \texttt{true}$ means target position j is present in the partial sequence $\hat{S}_t$ at time t. Rather than sampling each $\mathbf{m}_t^j$ independently, we construct $\mathbf{m}_t$ via an auxiliary space of Boolean variables $\mathbf{M} \in \{\texttt{true}, \texttt{false}\}^{y \times y}$, consisting of y independent CTMC processes, one per row, all initialized at $\mathbf{M}_0 = \texttt{false}$. The dynamics of row i are:

$$u_t(\mathbf{M}^{i,j} | \mathbf{M}_t^i) = \left(\lambda_t^{\text{indep}} \, \delta_{ij} + \mathbb{1}_{[\mathbf{M}_t^{i,j-1} \vee \mathbf{M}_t^{i,j+1}]} \, \lambda^{\text{prop}} \right) \left(\mathbb{1}_{[\mathbf{M}^{i,j}]} - \delta_{\mathbf{M}^{i,j}}(\mathbf{M}_t^{i,j}) \right). \tag{95}$$

The diagonal entry $\mathbf{M}^{i,i}$ switches to $\texttt{true}$ at the time-dependent rate $\lambda_t^{\text{indep}} = \dot{\kappa}_t / (1 - \kappa_t)$, independently of all other positions. Once $\mathbf{M}^{i,i}$ is active, neighboring off-diagonal entries $\mathbf{M}^{i,j}$ ($j \neq i$) switch to $\texttt{true}$ at a constant rate λ^{prop} whenever an adjacent entry ($\mathbf{M}_t^{i,j-1}$ or $\mathbf{M}_t^{i,j+1}$) is already $\texttt{true}$, propagating the $\texttt{true}$ state outward from position i. We use a cubic schedule $\kappa_t = t^3$ and set

$\lambda^{\text{prop}} = 3.0$. The per-position indicator is recovered by taking the column-wise disjunction:

$$\mathbf{m}_t^j = \mathbf{M}_t^{1,j} \vee \mathbf{M}_t^{2,j} \vee \cdots \vee \mathbf{M}_t^{y,j}. \tag{96}$$

When $\lambda^{\text{prop}} = 0$, each row activates only its diagonal entry, recovering the factorized probability path in which each $\mathbf{m}_t^j$ is independent.

Effective rate. When computing the training loss, the rate at which position j transitions from absent to present (i.e., $\mathbf{m}_t^j$ switches to `true`) is no longer the uniform $\dot{\kappa}_t/(1-\kappa_t)$ but an effective rate that depends on the local propagation state:

$$\lambda_{j,t}^{\text{eff}} = \lambda_t^{\text{indep}} + \sum_{l=1}^{y} \mathbb{1}\{\mathbf{M}_t^{l,j-1} \vee \mathbf{M}_t^{l,j+1}\}\, \lambda^{\text{prop}}. \tag{97}$$

Positions adjacent to already-present neighbors have a higher effective rate, while isolated positions receive only the independent rate.

Training loss. The localized path modifies both the Poisson NLL $\mathcal{L}_{\text{PP}}$ (Eq. 77) and the reconstruction loss $\mathcal{L}_{\text{rec}}$ (Eq. 78) by replacing the uniform weight $\dot{\kappa}_t/(1-\kappa_t)$ with the position-dependent effective rate. The modified Poisson NLL is:

$$\mathcal{L}_{\text{GP}}^{\text{loc}} = \mathbb{E}_{t,Y_1}\left[\sum_{i=0}^{X_t} \lambda_t^i - \sum_{i=0}^{X_t}\Big(\sum_{k\in\Omega_i} \lambda_{k,t}^{\text{eff}}\Big) \log \lambda_t^i\right], \tag{98}$$

and the modified reconstruction loss is:

$$\mathcal{L}_{\text{rec}}^{\text{loc}} = \mathbb{E}_{Y_1,Y_t}\left[-\sum_{i=0}^{X_t}\sum_{k\in\Omega_i} \lambda_{k,t}^{\text{eff}}\, \log \rho_\theta^i(s_k|Y_t)\right], \tag{99}$$

where the sum runs over the $X_1 - X_t$ not-yet-present positions k (grouped into bins Ω_i), and ρ_θ^i is the sampling distribution at insertion position i. The effective rate $\lambda_{k,t}^{\text{eff}}$ upweights reconstruction loss at positions adjacent to already-present residues, encouraging the model to extend locally consistent subsequences rather than predict isolated positions.

C.3 Peptide Co-Design

C.3.1 Data Specification

We use the dataset derived in PepFlow [34], which contains filtered PDB structure entries with peptide lengths ranging from 3 to 25. The full dataset contains 8,365 protein–peptide complex structures, which are clustered at 40% peptide sequence identity, yielding 292 unique clusters. Ten clusters across different lengths are selected as the test set, comprising 158 structures. The binding pocket is given as additional conditioning information, following the practice in PepFlow.

Each data pair is a complex of a target receptor and its bound peptide. Following PepFlow, the binding pocket is provided as a fixed conditioning context, and the peptide is the target for generation. The peptide is modeled at the all-atom level: a backbone frame (rotation and translation), the residue type, and up to four sidechain torsion angles per residue. During training, uniform sampling over the training clusters is performed.

C.3.2 Model Specification

Architecture We adapt PepFlow [34], an equivariant geometric GNN with heads to regress vector fields for all-atom peptide co-design. PepFlow represents all-atom structures with per-residue rotation and translation, amino acid logit, and up to four sidechain torsion angles. Built upon the IPA structure, PepFlow denoises these modalities together using (Riemannian) flow matching to co-design the peptide conditioned on the target protein structure. We enable variable-length generation with an order-preserving process: a rate head predicts how many residues to insert at each inter-residue gap, so the peptide length is generated rather than conditioned on the native length. Each inserted residue is predicted across all of PepFlow's modalities with additional heads when such an insertion happens.

Training We optimize GPFlow with AdamW (learning rate 10^{-4}, weight decay 0.01) at a total batch size of 48, keeping an EMA of the weights (decay 0.999) for evaluation. The model is trained across four NVIDIA RTX PRO 6000 GPUs for approximately two days. We adopt two techniques to improve the early rate prediction during generation. **Noisy context learning (NCL).** During training, the kept peptide context fed to the decoder is corrupted as $x_{\mathrm{ncl}} = w\, x_t + (1-w)\, \epsilon$, with $w \sim \mathcal{U}[0,1]$ and $\epsilon \sim \mathcal{N}(0, I)$, so the model learns to read structural guidance without relying on perfectly accurate context. **Scheduled sampling (SS).** With probability 0.5, the self-conditioning channel is fed the decoder's own clean-data prediction $x_{\mathrm{pred}} = x_t + (1-t)\, v_\theta(x_t, t, z)$ in place of the ground truth, exposing the model to its own output.

Sampling For each target, we generate 10 peptides over 250 sampling steps. The all-atom structure and sequence are produced with a low-temperature score-SDE over the continuous modalities (noise scale 0.05), while the peptide length is not supplied and emerges from the learned insertion rate.

C.3.3 Evaluation Pipeline

We use the metrics in PepFlow to evaluate the quality of generation across various modalities. For each baseline model, 10 samples are generated for each holdout target, and the metrics are averaged across all generations.

AAR Amino Acid Recovery (AAR) quantifies the sequence identity between the generated and the native peptide. AAR is computed as the overlap ratio between the generated and the native sequence. A higher AAR indicates a closer match between the generated and native peptides in terms of amino acid composition, reflecting the accuracy of the generated sequence.

RMSD To evaluate the generation quality of the peptides, the RMSD between the ground-truth and generated peptides is calculated from their CA coordinates using the Kabsch algorithm. A lower RMSD indicates a closer structural alignment between the generated and native peptides.

SSR The Secondary Structure Ratio (SSR) assesses the similarity between the secondary structure of the generated peptide and the ground-truth peptide. This metric is computed by determining the ratio of identical entries in the secondary structure labels of the two peptides. The secondary structure labels are obtained using the DSSP software. A higher SSR indicates a closer match in the secondary structure between the generated and native peptides.

BSR Binding Site Recovery (BSR) gauges the interaction similarity between the generated peptide–target pair and the native pair. BSR assesses whether the generated peptide recognizes residues in the target protein similarly to the native peptide. Following PepFlow, a residue is considered to be in the binding site if its CB atom is within a 6 Å radius of any peptide residue. BSR is calculated as the overlap between the binding sites derived from the generated and native peptides. A higher BSR indicates a closer match between the generated and native peptide-protein interactions.

Affinity The Affinity ratio represents the percentage of designed peptides exhibiting lower binding energy, indicating a higher binding affinity to the target protein compared to the native peptide. Following PepFlow, the binding energy is computed using the *InterfaceAnalyzerMover* in PyRosetta [11], after relaxing the complex and defining the interface between the peptide and target protein. A higher Affinity percentage indicates that the designed peptides exhibit enhanced binding affinity, suggesting potential improvements in their functional capabilities.

Stability The Stability ratio is defined as the proportion of designed peptides that exhibit a lower energy score compared to the native complex. Following PepFlow, we utilize the *FastRelax* method in PyRosetta, where each complex first undergoes relaxation, and the total score is evaluated using the REF2015 score function. The Stability metric is then calculated as the ratio of complexes with reduced energy scores, highlighting the designed peptides that contribute to the enhanced stability in the protein-peptide complex.

Designability Similar to previous structure design tasks, Designability assesses whether a generated peptide structure corresponds to a sequence that can fold into a structure similar to itself. Specifically,

we use ESMFold to predict the peptide structure and subsequently employ Rosetta *FlexPep* [41, 48] to dock the predicted structure into the target. The resulting docked structure is then compared with the native peptide structure, where a *designable* peptide should exhibit less than 2 Å RMSD to the native structure. Designability is then reported as the ratio of designable peptides across all test targets.

Diversity Similar to previous structure design tasks, Diversity is quantified by calculating all the pairwise TM-scores among the generated peptides for a given target using the original TM-align program [62]. The diversity metric is then defined as 1 minus the average TM-score. A higher diversity value indicates greater structural variation among the generated peptides, showcasing the extent of structural exploration in the design process.

Table 6: Per-subset breakdown of GPFlow's composite AAR/RMSD/SSR.

	Length-matched	Length-unmatched	Composite
AAR ↑ (%)	57.67	51.83	53.71
RMSD ↓ (Å)	0.94	1.60	1.39
SSR ↑ (%)	95.93	88.79	91.08

Adaptations to Variable-Length Design Among the metrics above, AAR, RMSD, and SSR are computed per residue and therefore require a correspondence between the generated peptide of length n and the native peptide of length m. When $n = m$, this is the identity correspondence $i \mapsto i$. Since GPFlow is a variable-length model, the generated peptide length n differs from m on 67.9% of samples, and no canonical identity correspondence exists. We therefore align the two peptides with an order-preserving map. When $n = m$, we use the identity correspondence; otherwise, we compute a Needleman-Wunsch alignment with zero gap cost, which yields a monotone matching of $\min(n, m)$ residue pairs. We average each metric over the matched residues only and exclude the $|n - m|$ unmatched residues, so the three metrics do not penalize the length difference, which is instead reported as a separate length statistic. The AAR, RMSD, and SSR computed under this scheme are reported in Table 6.

C.3.4 Baseline Information

In addition to the base PepFlow model, for which we use the official PepFlow checkpoint for evaluation with 200 flow sampling steps, we also compare GPFlow instantiation against two additional peptide co-design models or pipelines. We follow PepFlow's configurations for all three baselines for fair comparison.

RFDiffusion The same RFDiffusion [58] model used in motif scaffolding is also used here for conditional peptide design. The official RFDiffusion implementation is used with 200 diffusion steps. ProteinMPNN [16] is subsequently applied to redesign the amino acid types for evaluation.

ProteinGenerator ProteinGenerator [40] is built on RFDiffusion and additionally supports co-designing protein backbone structures and sequences. The official inference scripts are used with 200 diffusion steps. For a fair comparison, no additional hotspot or DSSP information is fed into the model.

D Ablation Studies

In this section, we ablate the key design choices of GPFlow on the unconditional structure generation task using the PDB dataset, reporting the same metrics as in Table 1.

D.1 Insertion Scheduler and Sampling Step Ablation

Table 7 presents an ablation study of the length-insertion scheduler κ_t and the number of sampling steps. Both the linear scheduler and the early-insertion scheduler ($\kappa_t := \min(t/0.3, 1)$, used in our main results) yield comparably high designability of 96.1, whereas the late-insertion scheduler

($\kappa_t := t^2$) noticeably degrades designability to 91.6. This suggests that growing the structure earlier in the generative trajectory leaves more steps for the continuous vector field to refine the inserted coordinates. For the number of sampling steps, increasing from 200 to 400 substantially improves designability (89.3 → 96.1), while 800 steps provide no further gain (95.7); we therefore adopt 400 steps as a balance between generation quality and sampling cost.

Table 7: Ablation studies of different GPFlow variants on the PDB dataset.

GPFlow Variant	Design-ability ↑	Diversity ↑	Novelty ↓		Struct% α/β
			PDB	AFDB	
400 steps, 0.3 early insertion	96.1	0.26	0.76	0.80	61.2/13.8
400 steps, late insertion (square)	91.6	0.26	0.75	0.79	59.2/14.2
200 steps	89.3	0.33	0.74	0.78	58.6/13.0
800 steps	95.7	0.22	0.76	0.80	60.1/14.3

D.2 Designability-Diversity Trade-Off

Table 8: Ablation studies of different sampling-noise scales γ for GPFlow on the PDB dataset.

γ	Design-ability ↑	Diversity ↑	Novelty ↓		Struct% α/β
			PDB	AFDB	
0.3	97.7	0.27	0.80	0.84	55.2/11.2
0.35 (default)	96.1	0.26	0.76	0.80	61.2/13.8
0.5	82.6	0.45	0.73	0.78	56.7/12.9

Table 8 reports the effect of the sampling noise scale γ in the SDE sampler (Eq. 81). A smaller γ mimics low-temperature sampling, increasing designability at the cost of diversity (e.g., $\gamma = 0.3$ attains 97.7 designability but only 0.27 diversity), whereas a larger $\gamma = 0.5$ reverses this trade-off (82.6 designability, 0.45 diversity). We adopt $\gamma = 0.35$ as the default, which balances high designability with reasonable diversity. This trade-off also explains the generally lower diversity of the Proteina-family models reported in Table 1.

D.3 τ-Leaping Ablation

Table 9: Ablation studies of τ-leaping for different sampling steps on unconditional structure design.

#Step	400		100	
	w/ τ	w/o τ	w/ τ	w/o τ
Length	153.7 ± 48.6	153.4 ± 43.5	162.1 ± 44.0	84.2 ± 18.3
Designability↑	**96.1%**	94.0%	83.6%	93.8%

The τ-leaping technique (Appendix B.5) is proposed to mitigate the large number of sampling steps required to achieve decent generation quality compared to standard flow-based models. For example, EditFlow requires 5k-10k sampling steps. For our structure design ablations in Table 9, 400 steps suffice for designability, with only a marginal improvement, likely due to the shorter length. However, switching to only 100 steps makes a crucial difference in recovering the original length distribution compared to the significantly shorter generations caused by rate overflow. The 93.8% designability is biased because the length distribution is significantly shorter.

D.4 Sequence Generation Technique Ablation

Table 10 demonstrates the impact of the sampling techniques of using the corrector and localized path in sequence generation. The corrector, which allows deletion of low-confidence amino acids, plays a

Table 10: Ablation studies of the corrector and localized path used in sequence modeling.

Techniques	Localized path		Factorized path	
	w/ corrector	w/o corrector	w/ corrector	w/o corrector
Length	479	513	519	525
pLDDT ↑	**68.7**	53.2	64.6	52.6

crucial role in improving pLDDT, whereas the localized path has a consistent but more moderate impact. In the factorized path results, the similar length shows that the pLDDT improvement is not due to a length-distribution discrepancy, but that the corrector is learning to remove inconsistent amino acids. The corrector has a more noticeable impact on the generation length for the localized path, reducing the average length by 34.

E Additional Results

In this section, we provide additional experimental results and visualizations of the GPFlow generations to further validate its performance and generation quality.

E.1 Unconditional Protein Generation

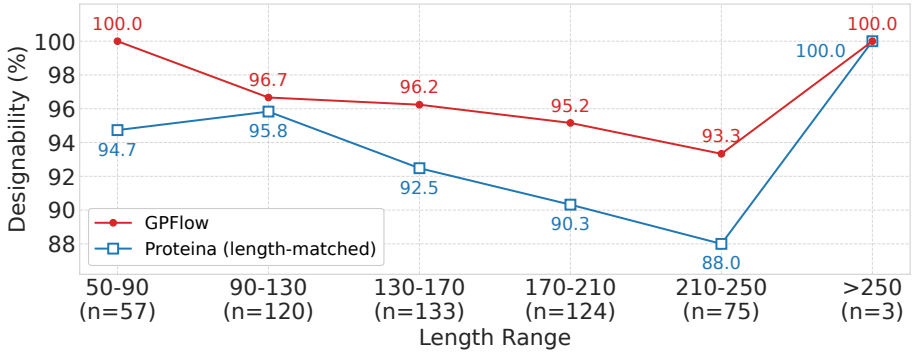


Figure 7: Designability per length range for generated structures for GPFlow and Proteina. Proteina generations are length-matched to GPFlow's generated length distribution.

For unconditional protein structure design, Figure 7 reports GPFlow's designability within each generated length bin. Additionally, we explicitly compare Proteina under a length-matched protocol: we specify each length so that Proteina generations' length distribution matches GPFlow's. Across all non-trivial matched length bins, GPFlow achieves better designability than the Proteina baseline, indicating that the improvement in the aggregate designability is not driven solely by the difference in the length range. Therefore, we treat this as strong evidence that the superior designability of GPFlow is not due to length mismatch but rather to its capture of the joint distribution over length and continuous coordinates.

E.2 Sequence-Based Length Control

In protein design settings where the desired length is known *a priori*, it would be useful to generate samples of a specified length. To this end, we evaluate the effectiveness of using classifier-free guidance (see Figure 8) for controlling protein length. Specifically, we generate 100 sequences at each target length from 100 to 1000 in increments of 100, using a guidance weight of $w = 0.75$ and 5,000 Euler steps. As shown in Figure 8, the generated lengths closely match the specified targets across the full range, demonstrating that GPFlow is capable of accurate and reliable protein length control.

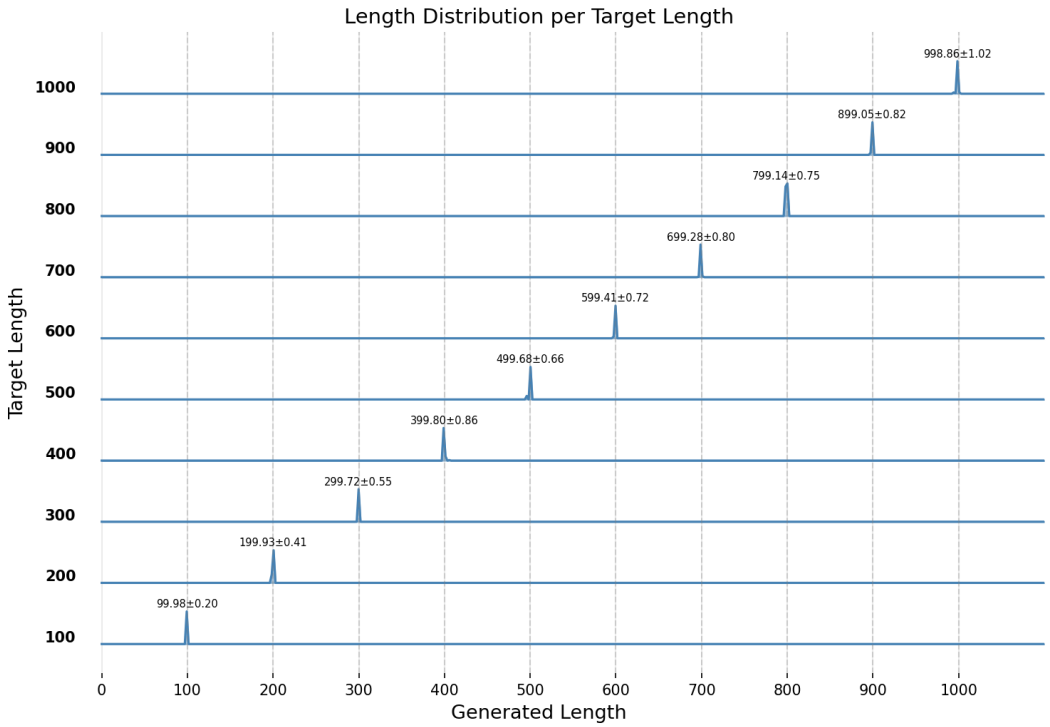


Figure 8: Evaluation of protein length control via classifier-free guidance. Each ridge shows the distribution of generated sequence lengths for a given target length, with the mean $\pm$ standard deviation annotated.

E.3 Structure-Based Motif Scaffolding

In Figure 9, we showcase the length distribution across all 16 motif tasks generated by GPFlow and Proteina. Compared to Proteina, our method yields a broader and smoother length distribution, reflecting its ability to adaptively generate backbones of appropriate lengths conditioned on the motif target, rather than relying on predefined contig templates. Remarkably, GPFlow can generate successful scaffolds longer than 200 residues, whereas Proteina generations are strictly limited by the contig templates, with most generations clustered around 75 residues.

We additionally demonstrate the per-task length distributions for representative tasks 1YCR, 3IXT, 5TPN, and 6E6R. For each task, the raw success lengths (red) and the average length within each unique success (blue) are plotted on the same scale in Figure 10. Because the sampling hyperparameters are identical across tasks, these results suggest that GPFlow adapts its generated lengths to the motif input, spanning a wider range for 3IXT while concentrating on a shorter-length peak for 6E6R. The longer unique successes observed for 3IXT and 5TPN further indicate that the model can produce successful and diverse scaffolds across a broad length range.

E.4 Sequence-Based Motif Scaffolding

We adapt our 642M sequence-generation GPFlow with the localized probability path described in Appendix C.2.5 to perform sequence-based motif scaffolding. During inference, generation starts from the motif tokens at $t = 0$, and the model grows a scaffold through progressive insertions with motif positions frozen. The learned deletion model serves as a corrector to refine the scaffold. The scaffold length emerges naturally from the learned rate function.

We compare against two sequence-based baselines: EvoDiff-OADM [1] and DPLM [57], using their official checkpoints and inference protocols. A design is successful if CA motif RMSD < 1Å and CA pLDDT > 70 as evaluated by ESMFold [38]. We generate 100 candidates per task across the 17

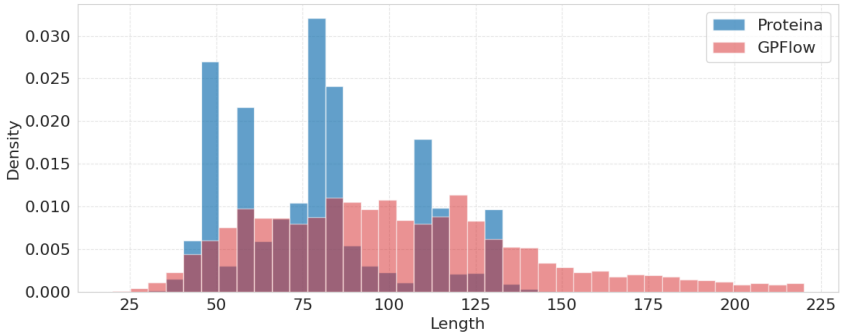


Figure 9: Aggregate protein length distribution across all 16 motif scaffolding tasks for GPFlow and Proteina.

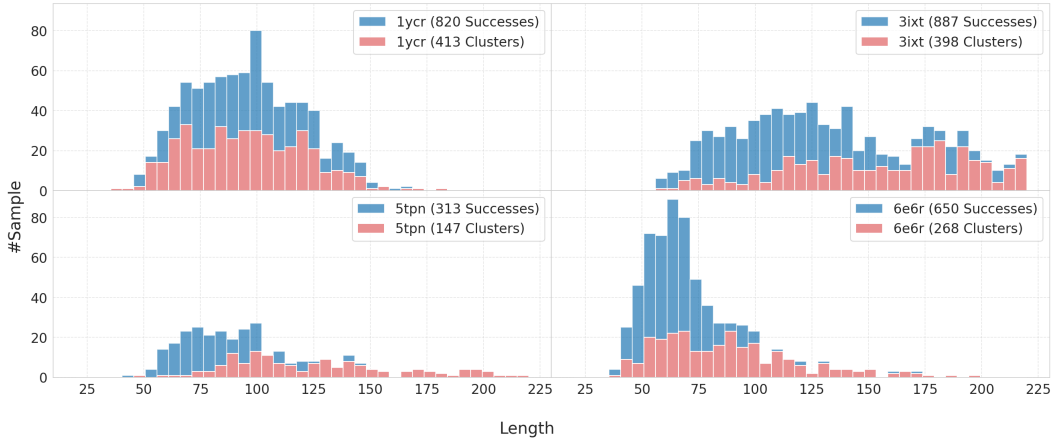


Figure 10: Protein length distributions for representative tasks for GPFlow. Raw success lengths and the average length within each unique success are plotted on the same scale.

motif problems adapted from the RFdiffusion benchmark [58] to the sequence setting by Alamdari et al. [1].

As shown in Table 11, GPFlow passes 10 out of 17 tasks, compared to 5/17 for EvoDiff and 6/17 for DPLM. Notably, GPFlow uniquely solves four tasks (5WN9, 2KL8, 4ZYP, 6VW1) where both baselines achieve zero success. On tasks where multiple methods succeed, GPFlow shows substantial gains: 87% on 3IXT versus 2% (EvoDiff) and 23% (DPLM); 59% on 7MRX versus 0% and 22%.

E.5 Peptide Co-Design

We compare the lengths of GPFlow-generated samples with those of native peptides. In Figure 11 (left), the generated and native length distributions agree closely. We further compare the generated length with the native length for each target (Figure 11, right). GPFlow recovers the lengths of short peptides nearly exactly; however, its conditional mean increasingly falls below the identity line as the native length increases, under-generating long peptides by approximately 1.6 residues for native lengths above 12. A possible contributing factor is the limited conditioning signal: GPFlow observes only the binding pocket and is not given the target peptide length. Thus, for longer peptides, residues extending beyond the pocket interface may be less directly constrained, potentially making their lengths harder to recover accurately.

F Broader Impacts

This work introduces Generalized Poisson Flow (GPFlow), a framework for variable-length generation evaluated on protein design. Its connection between generalized Poisson processes and

Table 11: Sequence-based zero-shot motif scaffolding success rates (100 generations per task). Superscripts denote the number of motif segments. Best results are shown in **bold**.

PDB	GPFlow (Ours)	DPLM [57]	EvoDiff [1]
1BCF^4	**0.31**	0.00	**0.31**
1PRW^2	0.05	**0.86**	0.31
1QJG^3	0.00	0.00	0.00
1YCR^1	0.24	**0.42**	0.01
2KL8^1	**0.01**	0.00	0.00
3IXT^1	**0.87**	0.23	0.02
4JHW^2	0.00	0.00	0.00
4ZYP^1	**0.01**	0.00	0.00
5IUS^2	0.00	0.00	0.00
5TPN^1	0.00	0.00	0.00
5TRV^1	0.00	0.00	0.00
5WN9^1	**0.13**	0.00	0.00
5YUI^3	0.00	0.00	0.00
6E6R^1	0.12	**0.90**	0.05
6EXZ^1	0.00	**0.02**	0.00
6VW1^2	**0.05**	0.00	0.00
7MRX^1	**0.59**	0.22	0.00
Pass rate	**10/17**	6/17	5/17

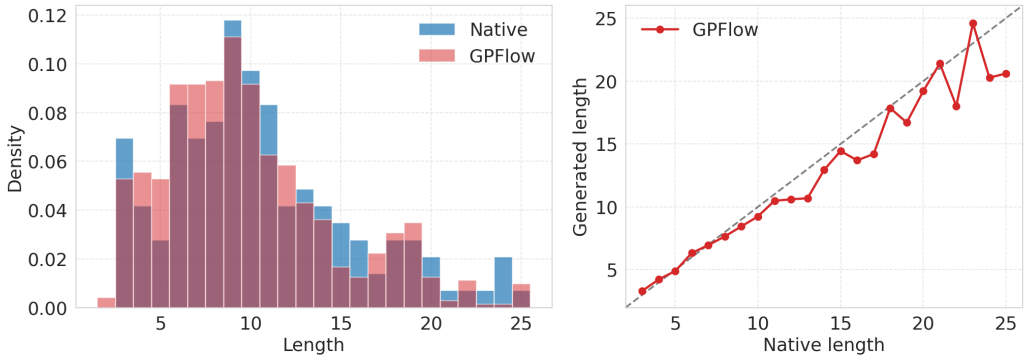


Figure 11: Peptide length. **Left**: length distributions of GPFlow-generated and native peptides. **Right**: mean generated length at each native length; the dashed line marks exact recovery.

probability flows may also be relevant to other domains with variable-dimensional data, although such applications are not evaluated here. In computational biology, GPFlow addresses the dependence of structural feasibility on sequence length in *de novo* protein design. Learning a distribution over backbone lengths may reduce manual length searches in motif-scaffolding applications relevant to vaccine and enzyme design. As with any advanced protein design technology, there is a potential for misuse, such as the design of harmful biological agents or toxins. The generalizability of our framework implies that it could potentially be applied to biological systems with unknown or hazardous properties. We are dedicated to ensuring the responsible use of our model for societal benefit.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We have clearly indicated our contributions and scope in the abstract and introduction.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We have discussed our limitation in Section 6.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: We have clearly provided the full set of assumptions together with complete proof in Appendix A for all the theorems and propositions.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We have disclosed all the information necessary to reproduce our experimental results in Appendix B and C, where we comprehensively describe our algorithm, loss choice, data pipeline, model architecture, and training and sampling hyperparameters.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The datasets used in this paper are openly and freely available. We will open-source our code once our paper gets accepted.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: We have specified all the training pseudocode in Appendix B. We have described our data specifications, model architecture, training hyperparameters, and sampling hyperparameters in Appendix C.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: Following existing work, the metrics we report are all collective ones aggregated over a large number of generations to mitigate the impact of stochasticity.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We have provided information on our computational resources in Appendix C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We conform, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We have discussed the potential societal impacts in Appendix F.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.

- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We have appropriately cited and credited the original authors, curators, and creators of the data, code, and models.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: This paper does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: This paper does not involve LLMs as any important, original, or non-standard components. We only used LLMs to help write the code, verify the mathematical correctness of the proof, and polish the paper.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.