GENERATOR MATCHING: GENERATIVE MODELING WITH ARBITRARY MARKOV PROCESSES

Peter Holderrieth[1,†], **Marton Havasi**[2], **Jason Yim**[1], **Neta Shaul**[2,3], **Itai Gat**[2],
Tommi Jaakkola[1], **Brian Karrer**[2], **Ricky T. Q. Chen**[2], **Yaron Lipman**[2]
[1]MIT CSAIL, [2]FAIR, Meta, [3]Weizmann Institute of Science
[†]Work done during an internship at FAIR, Meta.

ABSTRACT

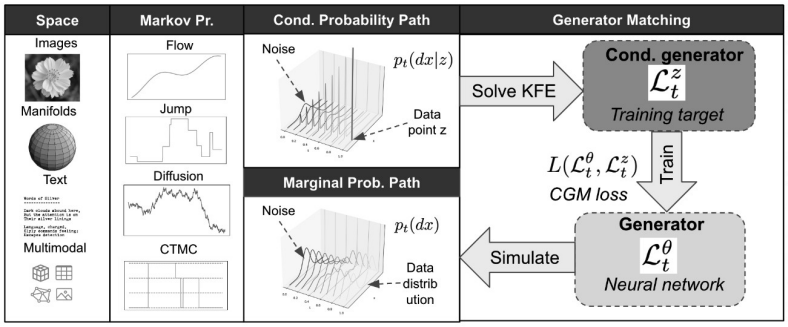
We introduce *Generator Matching*, a modality-agnostic framework for generative modeling using arbitrary Markov processes. Generators characterize the infinitesimal evolution of a Markov process, which we leverage for generative modeling in a similar vein to flow matching: we construct conditional generators which generate single data points, then learn to approximate the marginal generator which generates the full data distribution. We show that Generator Matching unifies various generative modeling methods, including diffusion models, flow matching and discrete diffusion models. Furthermore, it expands the design space to new and unexplored Markov processes such as jump processes. Finally, Generator Matching enables the construction of superpositions of Markov generative models and enables the construction of multimodal models in a rigorous manner. We empirically validate our method on image and multimodal generation, e.g. showing that superposition with a jump process improves performance.

1 INTRODUCTION

Early deep generative models—like VAEs (Kingma, 2013) and GANs (Goodfellow et al., 2014) generated samples in a single forward pass. With denoising diffusion models (DDMs) (Song et al., 2020; Ho et al., 2020), a paradigm shift happened were *step-wise* updates are used to transform noise into data. Similarly, scalable training of continuous normalizing flows (CNFs; Chen et al. 2018) via flow matching (Lipman et al., 2022; Liu et al., 2022; Albergo et al., 2023) allowed for high-quality and fast generative modeling by simulating an ODE. Since then, similar constructions based on diffusion and flows have also been applied to other modalities such as discrete data (Campbell et al., 2022; Gat et al., 2024) or data on manifolds (De Bortoli et al., 2022; Huang et al., 2022; Chen & Lipman, 2024) leading to a variety of models for different data types.

The single common property of the aforementioned generative models is their iterative step-wise nature: starting with a sample $X_0 \sim p_{\text{simple}}$ from an easy-to-sample distribution p_{simple}, they iteratively construct samples X_{t+h} of the next time step depending only on the current state X_t. Mathematically speaking, this means that they are all *Markov processes*. In this work, we develop a generative modeling framework that relies on that Markov property. At the core of our framework is the concept of a *generator* that describes the infinitesimal change of the distribution of a Markov process. We show that one can easily learn a *generator* through a family of scalable training objectives—a framework we coin *Generator Matching (GM)*.

Generator Matching unifies many existing generative modeling techniques across data modalities such as denoising diffusion models (Song et al., 2020), flow matching (Lipman et al., 2022), stochastic interpolants (Albergo et al., 2023), discrete diffusion models (Campbell et al., 2022; Gat et al., 2024; Lou et al., 2024a), among many others (see sec. 8). Most importantly, GM gives rise to new, unexplored models, and allows us to combine models across different classes of Markov processes. We make the following contributions:



Algorithm 1 Generator Matching recipe for constructing Markov generative model (theory in black, implementation in brown)

Step 0: Choose prior p_{simple}
Step 1: Choose $p_t(dx|z)$ such that marginal $p_0 = p_{\text{simple}}$ and $p_1 = p_{\text{data}}$
Step 2: Find solution $\mathcal{L}_t^z$ to KFE
Step 3: Choose Bregman div. as loss
Step 4: Construct neural net $\mathcal{L}_t^\theta$
Step 5: Minimize CGM loss using $\mathcal{L}_t^z$
Step 6: Sample using Algorithm 2.

Figure 1: Overview of the Generator Matching (GM) framework to construct generative models. GM works on any state space (including multi-modal) and Markov processes. *Flower image source:* `vecteezy.com`

1. **Generator Matching:** We present *Generator Matching* (GM), a framework for generative modeling with Markov processes on arbitrary state spaces. This framework unifies a diversity of prior generative modeling methods into a common framework that is modality-agnostic.
2. **Novel models:** On discrete and Euclidean spaces, we universally characterize the space of Markovian generative models identifying jump models as an unexplored model class for $\mathbb{R}^d$.
3. **Model combinations:** We show how GM allows to combine models in 2 ways: (1) *Markov superpositions* enable to construct ensembles of generative models; and (2) *Multimodal generative models* can be constructed by combining GM models built for single data modalities.
4. **Experiments:** On image and multimodal protein generation experiments, we show that jump models and Markov superpositions allow us to achieve competitive results.

2 Generative modeling via Probability Paths

Let S be a **state space**. Important examples are $S = \mathbb{R}^d$ (*e.g.*, images, vectors), S discrete (*e.g.*, language), S a Riemannian manifold (*e.g.*, geometric data) or their products for multimodal generation. In generative modeling, we are given samples $x_1, \dots, x_N \sim p_{\text{data}}$ from a distribution p_{data} on S and our goal is to generate novel samples $z \sim p_{\text{data}}$. GM works for arbitrary distributions, in particular those that do not have densities (*e.g.*, with discrete support). For general probability measures p, we use the notation $p(dx)$ where "dx" is a *symbolic* expression denoting integration with respect to p in a variable x. If for a distribution p a density exists with respect to a reference measure ν on S, we write $\frac{dp}{d\nu}(x)$ for its density.

A fundamental paradigm of recent state-of-the-art generative models is that they prespecify a transformation of a simple distribution p_{simple} (e.g. a Gaussian) into p_{data} via probability paths. Specifically, a **conditional probability path** is a set of time-varying probability distributions $(p_t(dx|z))_{0 \le t \le 1}$ depending on a data point $z \in S$. Together with the data distribution p_{data}, this induces a corresponding **marginal probability path** via the hierarchical sampling procedure:

$$z \sim p_{\text{data}}, x \sim p_t(dx|z) \quad \Rightarrow \quad x \sim p_t(dx) \tag{1}$$

i.e. first sample a data point $z \sim p_{\text{data}}$ and then sample $x \sim p_t(dx|z)$ from the conditional path. As we will see, this makes training scalable. The conditional probability path is usually chosen such that $p_0(dx|z) = p_{\text{simple}}$ and $p_1(dx|z) = \delta_z$ where δ_z is the Dirac delta distribution at z (i.e. the trivial, deterministic distribution returning z every draw). The associated marginal probability path interpolates between p_{simple} and p_{data}, leading to the first design principle of GM:

Principle 1: Given a data distribution p_{data}, choose a prior p_{simple} and a conditional probability path such that its marginal probability path $(p_t)_{0 \le t \le 1}$ fulfills $p_{\text{simple}} = p_0$ and $p_{\text{data}} = p_1$.

Two common constructions are mixtures (for arbitrary S) and geometric averages (for $S = \mathbb{R}^d$):

$$p_t(dx|z) = (1-\kappa_t)\cdot p_{\text{simple}}(dx) + \kappa_t \cdot \delta_z(dx) \quad \Leftrightarrow \quad x_t \sim \begin{cases} z & \text{with prob } \kappa_t \\ x_0 & \text{with prob } (1-\kappa_t) \end{cases} \qquad \blacktriangleright \text{ mixture} \tag{2}$$

$$p_t(dx|z) = \mathbb{E}_{x_0}\left[\delta_{\sigma_t x_0 + \alpha_t z}\right] \qquad \Leftrightarrow \quad x_t = \sigma_t x_0 + \alpha_t z \qquad \blacktriangleright \text{ geometric average} \tag{3}$$

where $x_t \sim p_t(\cdot|z), x_0 \sim p_{\text{simple}}, z \sim p_{\text{data}}$, and $\alpha_t, \sigma_t, \kappa_t \in \mathbb{R}_{\geq 0}$ are differentiable functions satisfying $\kappa_0 = \alpha_0 = \sigma_1 = 0$ and $\kappa_1 = \alpha_1 = \sigma_0 = 1$ and $0 \leq \kappa_t \leq 1$.

Remark. Note that the above includes specific ways of constructing probability paths, e.g. via a forward process in diffusion models (Song et al., 2020) or an interpolant function (Albergo et al., 2023) (see app. A.4). GM works for all these cases including if one conditions on more general latent variables z (Tong et al., 2023; Pooladian et al., 2023). Note also that in the diffusion literature, time is inverted ($t = 0$ corresponds to data).

3 Markov Processes

We briefly define time-continuous Markov processes, a fundamental concept in this work (Ethier & Kurtz, 2009). For $t \in [0,1]$, let $X_t \in S$ be a random variable. We call $(X_t)_{0\leq t\leq 1}$ a *Markov process* if it fulfills the following condition for all $0 \leq t_1 < t_2 < \cdots < t_n < t_{n+1} \leq 1$ and $A \subseteq S$ (measurable):

$$\mathbb{P}[X_{t_{n+1}} \in A | X_{t_1}, X_{t_2}, \ldots, X_{t_n}] = \mathbb{P}[X_{t_{n+1}} \in A | X_{t_n}] \qquad \blacktriangleright \text{ Markov assumption} \tag{4}$$

Informally, the above condition says that the process has no memory. If we know the present, knowing the past will not influence our prediction of the future. In table 1, we give an overview over important classes of Markov processes. Each Markov process has a **transition kernel** $(k_{t+h|t})_{0\leq t<t+h\leq 1}$ that assigns every $x \in S$ a probability distribution $k_{t+h|t}(\cdot|x)$ such that $\mathbb{P}[X_{t+h} \in A | X_t = x] = k_{t+h|t}(A|x)$. Due to the Markov assumption, there is a 1:1 correspondence between a Markov process and a transition kernel paired with an initial distribution p_0. We impose loose regularity assumptions on X_t listed in app. A.2.

In the context of GM, we use a Markov process as follows: Given a marginal path $(p_t(dx))_{0\leq t\leq 1}$ (see sec. 2), we want to train a model that allows to simulate a Markov process such that $X_0 \sim p_{\text{simple}} \Rightarrow X_t \sim p_t$ for all $0 \leq t \leq 1$. That is, if **initializing state at $t = 0$ with $X_0 \sim p_0$, the marginals of X_t will be** p_t for all $0 \leq t \leq 1$. Once we have found such a Markov process, we can simply generate samples from $p_1 = p_{\text{data}}$ by sampling $X_0 \sim p_0$ and simulating $X_{t+h} \sim k_{t+h|t}(\cdot|X_t)$ step-wise up to time $t = 1$. The challenge with such an approach is that an arbitrary general kernel $k_{t+h|t}$ is hard to parameterize in a neural network. One of the key insights in the development of diffusion models was that for small $h > 0$, the kernel $k_{t+h|t}$ can be closely approximated by a simple parametric distribution like Gaussians (Sohl-Dickstein et al., 2015; Ho et al., 2020). One can extend this idea to Markov processes leading to the concept of the *generator*.

4 Generators

Let us consider the transition kernel $k_{t+h|t}$ for small $h > 0$. Specifically, we consider an *informal* 1st-order Taylor approximation in t with an error term $o(h)$:

$$\text{“}k_{t+h|t} = k_{t|t} + h\mathcal{L}_t + o(h)\text{”}, \quad \mathcal{L}_t := \frac{d}{dh}\Big|_{h=0} k_{t+h|t}, \quad k_{t|t}(\cdot|x) = \delta_x \tag{5}$$

We call the 1st-order derivative $\mathcal{L}_t$ the *generator* of $k_{t+h|t}$ (Ethier & Kurtz, 2009; Rüschendorf et al., 2016). Similar to derivatives, generators are first-order *linear* approximations and, as we will see, easier to parameterize than $k_{t+h|t}$. Diffusion, flow, and other generative models can all be seen as algorithms to learn the generator of a Markov process (see table 1). However, as a probability measure is not a standard function, equation 5 is not well-defined yet. We will make it rigorous using *test functions*.

Test functions. Test functions are a way to “probe” a probability distribution. They serve as a theoretical tool to handle distributions as if they were real-valued functions. Specifically, we use a family $\mathcal{T}$ of bounded,

Name	Flow	Diffusion	Jump process	Continous-time Markov chain
Space S	$S = \mathbb{R}^d$	$S = \mathbb{R}^d$	S arbitrary	$\lvert S\rvert < \infty$
Parameters	$u_t(x) \in \mathbb{R}^d$	$\sigma_t^2(x) \in \mathbb{R}^{d\times d}$ σ_t^2 pos. semi-def.	Jump measure $Q_t(dy; x)$	$Q_t \in \mathbb{R}^{S\times S}$, $1^T Q_t = 0$ $Q_t(x'; x) \geq 0$ $(x' \neq x)$
Sampling	$X_{t+h} = X_t + hu_t(X_t)$	$X_{t+h} = X_t + \sqrt{h\sigma_t^2(X_t)}\epsilon_t$ $\epsilon_t \sim \mathcal{N}(0, I)$	$X_{t+h} = X_t$ with prob. $1 - h\int Q_t(dy; x)$ $X_{t+h} \sim \frac{Q_t(dy;x)}{\int Q_t(dy;x)}$ with prob. $h\int Q_t(dy; x)$	$X_{t+h} \sim (I + hQ_t)(\cdot; X_t)$
Generator $\mathcal{L}_t$	$\nabla f^T u_t$	$\frac{1}{2}\nabla^2 f \cdot \sigma_t^2$	$\int [f(y) - f(x)]Q_t(dy; x)$	$f^T Q_t^T$
KFE (Adjoint)	Continuity Equation: $\partial_t p_t = -\nabla \cdot [u_t p_t]$	Fokker-Planck Equation: $\partial_t p_t = \frac{1}{2}\nabla^2 \cdot [p_t \sigma_t^2]$	Jump Continuity Equation: $\partial_t \frac{dp_t}{d\nu}(x) =$ $\int Q_t(x; x')\frac{dp_t}{d\nu}(x') - Q_t(x'; x)\frac{dp_t}{d\nu}(x)v(dx')$	Mass preservation: $\partial_t p_t = Q_t p_t$
Marginal	$\mathbb{E}_{z\sim p_{1\lvert t}(\cdot\lvert x)}[u_t(x\lvert z)]$	$\mathbb{E}_{z\sim p_{1\lvert t}(\cdot\lvert x)}[\sigma_t^2(x\lvert z)]$	$\mathbb{E}_{z\sim p_{1\lvert t}(\cdot\lvert x)}[Q_t(dx'; x\lvert z)]$	$\mathbb{E}_{z\sim p_{1\lvert t}(\cdot\lvert x)}[Q_t(x'; x\lvert z)]$
CGM Loss (Example)	$\lVert u_t(x\lvert z) - u_t^\theta(x)\rVert^2$	$\lVert\sigma_t^2(x\lvert z) - [\sigma_t^\theta]^2(x)\rVert_2^2$	$(\int Q_t^\theta(x'; x)v(dx')$ $-Q_t(x'; x\lvert z)\log Q_t^\theta(x'; x)v(dx'))$	$(\sum_{x'\neq x} Q_t^\theta(x'; x)$ $-Q_t(x'; x\lvert z)\log Q_t^\theta(x'; x))$

Table 1: Examples of Markov models that can be learnt with GM. Derivations are in app. A.5. For diffusion, we assume zero drift. KFE is listed in its adjoint version, i.e. assumes jump kernel $Q_t(y; x)$ and density $\frac{dp_t}{d\nu}(x)$ exists with respect to reference ν. For Lebesgue measure ν, we write $\frac{dp_t}{d\nu}(x) = p_t(x)$.

integrable functions $f : S \to \mathbb{R}$ that *characterize* probability distributions fully, *i.e.*, two probability distributions μ_1, μ_2 are equal if and only if $\mathbb{E}_{x\sim\mu_1}[f(x)] = \mathbb{E}_{x\sim\mu_2}[f(x)]$ for all $f \in \mathcal{T}$. Generally speaking, one chooses $\mathcal{T}$ to be as "nice" (or regular) as possible. For example, if $S = \mathbb{R}^d$, the space $\mathcal{T} = C_c^\infty$ of infinitely differentiable functions with compact support fulfills that property. We define the action of the marginal p_t and transition kernels $k_{t+h|t}$ for all $f \in \mathcal{T}$ via the linear function defined via

$$\langle p_t, f\rangle \overset{\text{def}}{=} \int f(x)p_t(dx) = \mathbb{E}_{x\sim p_t}[f(x)] \qquad \blacktriangleright \text{ marginal action} \tag{6}$$

$$\left\langle k_{t+h|t}, f\right\rangle(x) \overset{\text{def}}{=} \left\langle k_{t+h|t}(\cdot|x), f\right\rangle = \mathbb{E}\left[f(X_{t+h})|X_t = x\right] \qquad \blacktriangleright \text{ transition action} \tag{7}$$

where the marginal action maps each test function f to a scalar $\langle p_t, f\rangle \in \mathbb{R}$, while the transition action maps a real-valued function $x \mapsto f(x)$ to a another real-valued function $x \mapsto \left\langle k_{t+h|t}, f\right\rangle(x)$. The tower property implies that $\left\langle p_t, \left\langle k_{t+h|t}, f\right\rangle\right\rangle = \langle p_{t+h}, f\rangle$. We note that the above is only a "symbolic" dot product but becomes a "proper" dot product if a density $\frac{dp_t}{d\nu}$ exists, i.e. $\langle p_t, f\rangle = \int f(x)\frac{dp_t}{d\nu}(x)\nu(dx)$.

Generator definition. Let us revisit equation 5 and define the derivative of $k_{t+h|t}$. With the test function perspective in mind, we can take derivatives of $\left\langle k_{t+h|t}, f\right\rangle(x)$ per $x \in S$ and define

$$\frac{d}{dh}\Big|_{h=0}\left\langle k_{t+h|t}, f\right\rangle(x) = \lim_{h\to 0}\frac{\left\langle k_{t+h|t}, f\right\rangle(x) - f(x)}{h} \overset{\text{def}}{=} [\mathcal{L}_t f](x). \tag{8}$$

We call this action the *generator* $\mathcal{L}_t$ (and define it for all f for which the limit exists uniformly in x and t, see app. A.1). In table 1, there are several examples of generators listed with derivations in app. A.5. With this definition, the Taylor series in equation 5 has the, now well-defined, form as $\left\langle k_{t+h|t}, f\right\rangle = f + h\mathcal{L}_t f + o(h)$.

Under mild regularity assumptions, there is a unique correspondence between the generator and the Markov process (see (Ethier & Kurtz, 2009; Pazy, 2012)). This allows us to parameterize a Markov process:

Principle 2: Parameterize a Markov process via a parameterized generator $\mathcal{L}_t^\theta$.

Of course, it is hard to parameterize a linear operator $\mathcal{L}_t$ on function spaces directly via a neural network. A simple solution is to restrict ourselves to certain subclasses of Markov processes and parameterize it linearly with a neural network (see app. A.6 for details and examples). For example, flow matching restricts itself to generators of the form $\mathcal{L}_t f = \nabla f(x)^T u_t^\theta(x)$ which correspond to flows. However, as we will show now, we can in fact fully characterize generators on specific spaces.

Theorem 1 (Universal characterization of generators)**.** *Under regularity assumptions (see app. A.2), the generators of a Markov processes X_t ($0 \le t \le 1$) take the form:*

1. ***Discrete** $|S| < \infty$: The generator is given by a rate transition matrix Q_t and the Markov process corresponds to a continuous-time Markov chain (CTMC).*
2. ***Euclidean space** $S = \mathbb{R}^d$: The generator has a representation as a sum of components described in table 1, i.e.,*

$$\mathcal{L}_t f(x) = \underbrace{\nabla f(x)^T u_t(x)}_{\textit{flow}} + \underbrace{\frac{1}{2}\nabla^2 f(x) \cdot \sigma_t^2(x)}_{\textit{diffusion}} + \underbrace{\int [f(y) - f(x)]\, Q_t(dy; x)}_{\textit{jump}} \tag{9}$$

*where $u : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$ is a **velocity field**, $\sigma : [0,1] \times \mathbb{R}^d \to S_d^{++}$ the **diffusion coefficient** (S_d^{++}=positive semi-definite matrices), and $Q_t(A|x)$ is a finite measure called **jump measure**. $\nabla^2 f(x)$ describes the Hessian of f and $\nabla^2 f(x) \cdot \sigma_t^2(x)$ describes the Frobenius inner product.*

The proof adapts a known result in the mathematical literature and can be found in app. C.1. This result allows us to not only characterize a wide class of Markov process models but to characterize the design space *exhaustively* for $S = \mathbb{R}^d$ or S discrete. In Euclidean space, people have considered learning the flow parts of the generator and for diffusion models, using a fixed σ_t for a diffusion. Learning σ_t or jump models on non-discrete spaces have not (or rarely) been considered. A general recipe to sample from a Markov process with a universal generator is presented in alg. 2. For $S = \mathbb{R}^d$, we can therefore simplify Principle 2:

Principle 2 ($S = \mathbb{R}^d$): Parameterize a Markov process (*e.g.*, using a neural network) via a generator $\mathcal{L}_t$ that is composed of (a subset of) velocity u_t, diffusion coefficient σ_t^2, and jump measure Q_t.

5 Kolmogorov Forward Equation and Marginal Generator

Beyond parameterizing a Markov process, the generator has a further use-case in the Generator Matching framework: checking if a Markov process generates a desired probability path p_t. We discuss the latter now using the **Kolmogorov Forward Equation (KFE)**. Specifically, the evolution of the marginal probabilities p_t of a Markov process X_t are governed by the generator $\mathcal{L}_t$, as can be seen by computing:

$$\partial_t \langle p_t, f\rangle = \frac{d}{dh}\Big|_{h=0} \langle p_{t+h}, f\rangle = \left\langle p_t, \frac{d}{dh}\Big|_{h=0} \langle k_{t+h|t}, f\rangle \right\rangle \overset{(8)}{=} \langle p_t, \mathcal{L}_t f\rangle \tag{10}$$

where we used that the $\langle p_t, \cdot\rangle$ operation is linear to swap the derivative, and the fact that $\langle p_t, \langle k_{t+h|t}, f\rangle\rangle = \langle p_{t+h}, f\rangle$. This shows that given a generator $\mathcal{L}_t$ of a Markov process X_t we can recover its marginal probabilities via their infinitesimal change,

$$\partial_t \langle p_t, f\rangle = \langle p_t, \mathcal{L}_t f\rangle \qquad \blacktriangleright \text{ Kolmogorov Forward Equation (KFE)} \tag{11}$$

Conversely, if a generator $\mathcal{L}_t$ of a Markov process X_t satisfies the above equation, then X_t generates the probability path $(p_t)_{0\le t\le 1}$, i.e. initializing $X_0 \sim p_0$ will imply that $X_t \sim p_t$ for all $0 \le t \le 1$ (see app. A.2) (Rogers & Williams, 2000). Therefore, the key challenge of Generator Matching is:

Principle 3*: Given a marginal probability path $(p_t)_{0\le t\le 1}$, find a generator satisfying the KFE.

Remark - Adjoint KFE. We note that the above version of the KFE determines the evolution of expectations of test functions f. Whenever a probability density $\frac{dp_t}{d\nu}(x)$ exists, one can use the *adjoint KFE* (see table 1 for examples and app. A.3). In this form, the KFE generalizes many equations used to develop generative models such as the Fokker-Planck or the continuity equation (Song et al., 2020; Lipman et al., 2022).

We now show how to find a generator that generates a marginal probability path p_t with conditional path $p_t(\cdot|z)$. Assume that for every data point $z \in S$, we found a generator $\mathcal{L}_t^z$ that generates $p_t(\cdot|z)$. We call $\mathcal{L}_t^z$ **conditional generator**. This allows us to construct a generator for the marginal path (**marginal generator**):

Proposition 1. *The marginal probability path $(p_t)_{0\leq t\leq 1}$ is generated by a Markov process X_t with generator*

$$\mathcal{L}_t f(x) = \mathbb{E}_{z\sim p_{1|t}(\cdot|x)}[\mathcal{L}_t^z f(x)] \tag{12}$$

where $p_{1|t}(dz|x)$ is the posterior distribution (i.e. the conditional distribution over data z given an observation x). For $S = \mathbb{R}^d$ and the representation in eq. (9)*, we get a marginal representation of $\mathcal{L}_t f(x)$ given by:*

$$\nabla f(x)^T \mathbb{E}_{z\sim p_{1|t}(\cdot|x)}[u_t(x|z)] + \frac{\nabla^2 f(x)}{2} \cdot \mathbb{E}_{z\sim p_{1|t}(\cdot|x)}[\sigma_t^2(x|z)] + \int [f(y) - f(x)]\, \mathbb{E}_{z\sim p_{1|t}(\cdot|x)}[Q_t(dy; x|z)]$$

Generally, an identity as in eq. (12) *holds for any linear parameterization of the generator (see app. A.6).*

The proof relies on the linearity of the KFE (see app. C.2). Proposition 1 immensely simplifies the construction of a Markov process that generates a desired probability path. To find the right training target, we only need to find a solution for the KFE for the conditional path. This simplifies Principle 3* to:

Principle 3: Derive a *conditional* generator $\mathcal{L}_t^z$ satisfying the KFE for the conditional path $p_t(\cdot|z)$.

In denoising diffusion models, the strategy to find solutions to a KFE is to construct a probability path via a forward noising process and then use a time-reversal of that process as a solution to the KFE (we illustrate this in app. H.2). Here, we illustrate two novel solutions for the KFE for common conditional probability paths on $\mathbb{R}^d$ in fig. 2. We discuss them here for $d = 1$ (in sec. 7.2 it is discussed how to easily extend it to $d > 1$).

Example 1 - Jump solution to geometric average. Current state-of-the-art models use a geometric average probability path given by $p_t(\cdot|z) = \mathcal{N}(tz, (1-t)^2)$ called CondOT path (Lipman et al., 2022). We ask the question: are there other Markov processes that follow the same probability path? As derived in app. E.7, another solution is given by a jump model with rate kernel $Q_t : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$:

$$Q_t(x'; x|z) = \frac{[k_t(x)]_+ [-k_t(x')]_+ p_t(x'|z)}{(1-t)^3 \int [-k_t(\tilde{x})]_+ p_t(\tilde{x}|z) d\tilde{x}}, \quad k_t(x) = x^2 - (t+1)xz - (1-t)^2 + tz^2 \tag{13}$$

where $[x]_+ := \max(x, 0)$. In fig. 2, we illustrate how a jump model trained with this conditional rate has the same *marginal probability* path as common flow models but with significantly different *sample* paths.

Example 2 - Pure diffusion solution to mixture path. GM allows to learn the diffusion coefficient σ_t^2 of an SDE. We illustrate this for the mixture path $p_t(dx|z) = \kappa_t \delta_z + (1-\kappa_t)\text{Unif}_{[a_1,a_2]}$. We introduce a solution that we call **"pure diffusion"** (see app. E.3). The corresponding Markov process is given by an SDE with no drift (*i.e.*, no vector field) and diffusion coefficient given by

$$\sigma_t^2(x|z) = 2\dot{\kappa}_t \frac{a_2 - a_1}{1 - \kappa_t} \left(\frac{1}{2} \frac{(z-a_1)^2}{a_2 - a_1} + [x - z]_+ - \frac{1}{2} \frac{(x-a_1)^2}{a_2 - a_1}) \right) \tag{14}$$

We add an additional reflection term at the boundaries of the data support (see app. E.3). Note the striking feature of this model: It only specifies how much noise to add to the current state. Still, it is able to generate data (see fig. 2). This is strictly different than "denoising diffusion models" because they *corrupt* data (as opposed to *generating*) with a diffusion process and their $\sigma_t^2(x) = \sigma_t^2$ is state-independent and usually fixed.

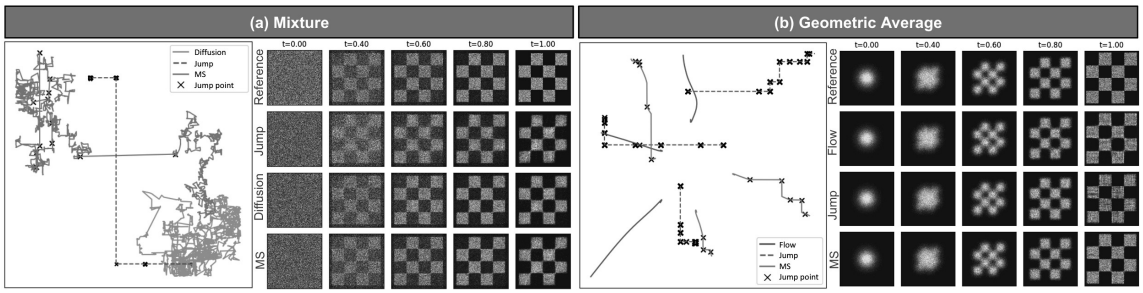


Figure 2: Illustration of Markov models trained with different KFE solutions for the same probability path. The paths for individual samples are plotted *across* time in one plot. 2d histograms of generated samples are plotted per time point. Although the individual sample paths look very different, the marginal probability path (histogram) are the same up to approximation error (Geometric average $\sim$ example 1, mixture $\sim$ example 2).

6 Generator Matching

We now discuss how to train a parameterized generator $\mathcal{L}_t^\theta$ to approximate the "true" marginal generator $\mathcal{L}_t$. In practice, $\mathcal{L}_t^\theta$ is linearly parameterized by a neural network $F_t^\theta : S \times [0,1] \to \Omega$ where $\Omega \subset V$ is a convex subset of some vector space V with inner product $\langle \cdot, \cdot \rangle$ (see app. A.6 for details). Our goal is to approximate the ground truth parameterization $F_t : S \times [0,1] \to \Omega$ of $\mathcal{L}_t$. For example, $F_t = u_t$ for flows, $F_t = \sigma_t^2$ for diffusion, or $F_t = Q_t$ for jumps (see table 1). We train the neural network F_t^θ to approximate F_t. As a distance function on Ω, we consider **Bregman divergences** defined via a convex function $\phi : \Omega \to \mathbb{R}$ as

$$D(a,b) = \phi(a) - [\phi(b) + \langle a - b, \nabla\phi(b) \rangle], \quad a, b \in \Omega \tag{15}$$

which are a general class of loss functions including many examples such as MSE or the KL-divergence (see app. C.3.1). We use D to measure how well F_t^θ approximates F_t via the **Generator Matching loss** defined as

$$L_{\text{gm}}(\theta) \stackrel{\text{def}}{=} \mathbb{E}_{t\sim\text{Unif}, x\sim p_t}\left[D(F_t(x), F_t^\theta(x))\right] \qquad \blacktriangleright \text{ Generator Matching} \tag{16}$$

Unfortunately, the above training objective is intractable as we do not know the marginal generator $\mathcal{L}_t$ and also no parameterization F_t of the marginal generator. To make training tractable, let us set F_t^z to be a linear parameterization of the conditional generator $\mathcal{L}_t^z$ with data point z (see app. A.6). For clarity, we reiterate that by construction, we know $F_t^\theta, F_t^z, p_t(\cdot|z), D$ as well as can draw data samples $z \sim p_{\text{data}}$ but the shape of F_t is unknown. By proposition 1, we can assume that F_t has the shape $F_t(x) = \int F_t^z(x) p_{1|t}(dz|x)$. This enables us to define the **conditional Generator Matching loss** as

$$L_{\text{cgm}}(\theta) \stackrel{\text{def}}{=} \mathbb{E}_{t\sim\text{Unif}, z\sim p_{\text{data}}, x\sim p_t(\cdot|z)}\left[D(F_t^z(x), F_t^\theta(x))\right] \qquad \blacktriangleright \text{ Conditional Generator Matching} \tag{17}$$

This objective is tractable and scalable. It turns out that we can use it to minimize the desired objective.

Proposition 2. *For any Bregman divergence, the GM loss L_{gm} has the same gradients as the CGM loss L_{cgm}, i.e. $\nabla_\theta L_{gm}(\theta) = \nabla_\theta L_{cgm}(\theta)$. Therefore, minimizing the CGM loss with Stochastic Gradient Descent will also minimize the GM loss. Further, for this property to hold, D must necessarily be a Bregman divergence.*

See app. C.3 for a proof. Note the significance of proposition 2: we can learn $\mathcal{L}_t$ with a scalable objective. Further, we can **universally characterize the space of loss functions**, including unexplored and new loss functions for diffusion models. In table 1, we list examples of several CGM loss functions. Often it is also possible to derive losses that give lower bounds on the model log-likelihood (ELBO bounds, see app. D).

Principle 4: Train $\mathcal{L}_t^\theta$ by minimizing the CGM loss with a Bregman divergence.

With this, we arrived at the last principle of GM. In alg. 1, we summarize the Generator Matching recipe for constructing generative models.

7 APPLICATIONS OF GENERATIVE MATCHING THEORY

GM provides a unifying framework for many existing generative models (see sec. 8), as well as gives rise to new models. Beyond that, the generality of GM in itself has several use cases that we discuss in this section.

7.1 COMBINING MODELS

The generator is a linear operator and the KFE $\partial_t \langle p_t, f \rangle = \langle p_t, \mathcal{L}_t f \rangle$ is a linear equation. These two properties enable us to combine generative models for the same state space S in different ways.

Proposition 3 (Combining models)**.** *Let p_t be a marginal probability path, then the following generators solve the KFE for p_t and consequently define a generative model with p_t as marginal:*

1. ***Markov superposition:*** *$\alpha_t^1 \mathcal{L}_t + \alpha_t^2 \mathcal{L}'_t$, where $\mathcal{L}_t, \mathcal{L}'_t$ are two generators of Markov processes solving the KFE for p_t, and $\alpha_t^1, \alpha_t^2 \geq 0$ satisfy $\alpha_t^1 + \alpha_t^2 = 1$. We call this a **Markov superposition**.*
2. ***Divergence-free components:*** *$\mathcal{L}_t + \beta_t \mathcal{L}_t^{div}$, where $\mathcal{L}_t^{div}$ is a generator such that $\left\langle p_t, \mathcal{L}_t^{div} f \right\rangle = 0$ for all $f \in \mathcal{T}$, and $\beta_t \geq 0$. We call such $\mathcal{L}_t^{div}$ **divergence-free**.*
3. ***Predictor-corrector:*** *$\alpha_t^1 \mathcal{L}_t + \alpha_t^2 \bar{\mathcal{L}}_t$, where $\mathcal{L}_t$ is a generator solving the KFE for p_t in forward-time and $\bar{\mathcal{L}}_t$ is a generator solving the KFE in backward time, and $\alpha_t^1, \alpha_t^2 \geq 0$ with $\alpha_t^1 - \alpha_t^2 = 1$.*

A proof can be found in app. C.4. Markov superpositions can be used to combine generative models of different classes, *e.g.*, one could combine a flow and a jump model. These can be 2 networks trained separately or we can train two models in one network simultaneously. We illustrate Markov superpositions in fig. 2. To find divergence-free components, one can use existing Markov-Chain Monte-Carlo (MCMC) algorithms - such as Hamiltonian Monte Carlo, Langevin dynamics, or approaches based on detailed balance - all of these algorithms are general recipes to find divergence-free components.

7.2 MULTIMODAL AND HIGH-DIMENSIONAL GENERATIVE MODELING

GM allows us to easily combine generative models from two state spaces S_1, S_2 into the product space $S_1 \times S_2$ in a rigorous, principled, and simple manner. This has two advantages: (1) we can design a joint multi-modal generative model easily and (2) we can often reduce solving the KFE in high dimensions to the one-dimensional case. We state here the construction informally and provide a rigorous treatment in app. C.5.

Proposition 4 (Multimodal generative models - Informal version)**.** *Let $q_t^1(\cdot|z_1), q_t^2(\cdot|z_2)$ be two conditional probability paths on state spaces S_1, S_2. Define the conditional factorized path on $S_1 \times S_2$ as $p_t(\cdot|z_1, z_2) = q_t^1(\cdot|z_1) q_t^2(\cdot|z_2)$. Let $p_t(dx)$ be its marginal path.*

1. ***Conditional generator:*** *To find a solution to the KFE for the conditional factorized path, we only have to find solutions to the KFE for each S_1, S_2. We can combine them component-wise.*
2. ***Marginal generator:*** *The marginal generator of $p_t(dx)$ can be parameterized as follows: (1) parameterize a generator on each S_i but make it values depend on all dimensions; (2) During sampling, update each component independently as one would do for each S_i in the unimodal case.*
3. ***Loss function:*** *We can simply take the sum of loss functions for each S_i.*

As a concrete example, let us consider joint image-text generation with a joint flow and discrete Markov model with $S_1 = \mathbb{R}^d, S_2 = \{1, \dots, N\}$. To build a multimodal model, we can simply make the vector field $u_t(x_t^1, x_t^2) \in \mathbb{R}^d$ depend on both modalities x_t^1, x_t^2 but update the flow part via $X_{t+h}^1 = X_t^1 + h u_t(X_t^1, X_t^2)$. Similarly, the text updates depend on both (X_t^1, X_t^2). In app. F, we give another example for jump models.

8 RELATED WORK

GM unifies a diversity of previous generative modeling approaches. We discuss here a selection for $S = \mathbb{R}^d$ and S discrete. App. H includes an extended overview and models for other S (e.g. manifolds, multimodal).

Denoising Diffusion and Flows in $\mathbb{R}^d$. From the perspective of GM, a "denoising diffusion model" is a flow model that is learnt using the CGM loss with the mean squared error. During sampling, a divergence-free component given via Langevin dynamics (Flow + SDE) can be be added for stochastic sampling (see proposition 3). If we set the weight of that component to 0, we recover the probability flow ODE (Song et al., 2020). To the best of our knowledge, it has not been explored in the literature yet whether one could *learn* a state-dependent diffusion coefficient $\sigma_t(x)$ as opposed to fixing it. Our framework allows for that as we illustrate in fig. 2. Flow matching and rectified flows (Lipman et al., 2022; Liu et al., 2022) are immediate instances of Generator Matching leveraging the flow-specific versions of the KFE given by the continuity equation (see table 1). Stochastic interpolants (Albergo et al., 2023) extend general flow-based models by learning an additional divergence-free Langevin dynamics component separately (see proposition 3 (b)) and showcase the advantages of adding it both theoretically and practically.

Discrete models and LLMs. In discrete spaces, Generator Matching recovers generative modeling via continuous-time Markov chains (Campbell et al., 2022; Santos et al., 2023; Gat et al., 2024), often coined "discrete diffusion models". These models use a version of proposition 4 using factorized probability paths to make the generator (=rate matrix Q_t) update each dimension independently. SEDD (Lou et al., 2024b) use the same Bregman divergence but with a different linear parameterization of the generator, namely via the discrete score. Theoretically, by using auto-regressive probability paths and fixing the jump times via high jump intensities, one can also recover common language model training as an edge case of GM.

Markov generative modeling. The most closely-related work to ours is Benton et al. (2024) that focus on recovering existing denoising diffusion models into a common framework. Here, we try to fully characterize the design space of Markov generative models as a whole and identify novel parts - *e.g.*, by introducing jump models on $\mathbb{R}^d$, Markov superpositions, universal characterizations of the space of generators, novel solutions to the KFE, Bregman divergence losses as the natural loss classes, among others.

9 Experiments

The design space of the GM framework is extraordinarily large. At the same time, single classes of models (*e.g.*, diffusion and flows) have already been optimized over many previous works. Therefore, we choose to focus on 3 aspects of GM: (i) Jump models as a novel class of models (ii) The ability of combining different model classes into a single generative model (iii) The ability to design models for multiple data modalities.

New models - Jump model. We first study jump models as a novel model class in Euclidean space. We use the jump model defined in eq. (13) and extend it to multiple dimensions using proposition 4. The jump kernel is parameterized with a U-Net architecture (see app. F for details). We use the loss from table 1. In app. D, we show that this corresponds to an ELBO loss. We apply the model on CIFAR10 and the ImageNet32 (blurred faces) datasets. As jump models do not have yet an equivalent of classifier-free guidance, we focus on unconditional generation. A challenge for a fair comparison is that flow models can use higher-order ODE samplers, while sampling for jump models in $\mathbb{R}^d$ is only done with Euler sampling so far. Hence, we ablate over this choice. As one can see in fig. 4, the jump model can generate realistic images of high quality. In table 2, we show quantitative results. While lacking behind current state-of-the art models, the jump model shows very promising results as a first version of an unexplored class of models.

Method	CIFAR10	ImageNet
DDPM (Ho et al., 2020)	3.17	6.99
VP-SDE (Song et al., 2020)	3.01	6.84
EDM (Karras et al., 2022)	1.98	–
Flow model (Euler)	2.94	4.58
Jump model (Euler)	4.23	7.66
Jump + Flow MS (Euler)	**2.49**	**3.47**
Flow model (2nd order)	2.48	3.59
Jump + Flow MS (mixed)	**2.36**	**3.33**

Table 2: Experimental results for image generation. FID scores are listed. MS=Markov superposition. Euler: euler sampling. 2nd order: 2nd order ODE sampler. Mixed: Flow uses 2nd order sampler and jump uses Euler sampling.

Combining models - Markov superposition. Next, we train a flow and jump model in the same architecture. We validate that the flow part achieves the state-of-the-art results as before. We then combine both models via a Markov superposition. As one can see in table 2, a Markov superposition of a flow and jump model boosts the performance of each other. For Euler sampling, we see significant improvements. We can also combine 2nd order samplers for flows with Euler sampling for jumps in a "mixed" sampling method (see table 2) leading to improvements of the current SOTA by flow models. We anticipate that with further improvements of the jump model, the increased performance via Markov superposition will be even more pronounced.

Multimodal state spaces - Protein experiments. GM allows us to design models for arbitrary and complex state spaces. To illustrate this, we show that GM allows to easily improve *MultiFlow*, a state-of-the-art model for joint protein structure and amino acid sequence generation (Campbell et al., 2024b), without even re-training the model. Specifically, we derive a novel solution $\mathcal{L}_t^z$ to the KFE on $S = SO(3)$ with a jump model (app. G.1). We then use proposition 4 to make it multi-dimensional and combine it with a flow model on $\mathbb{R}^d$ and a discrete Markov model on $\{1, \ldots, n\}^d$ for $n = 20$ (# amino acids), i.e. the state space becomes $S = \mathbb{R}^d \times SO(3)^d \times \{1, \ldots, 20\}^d$. Using the pre-trained MultiFlow without any fine-tuning, we "pseudo-marginalize" the conditional jumps by predicting $x_1 \in SE(3)$ and then taking a conditional step with $\mathcal{L}_t^z$. In fig. 3, we can see examples of generated proteins. We benchmark our results following Yim et al. (2023a). Table 3 shows our results of incorporating jumps with MultiFlow compared to baselines. In multimodal setting, both sequence and structure state spaces are sampled jointly while in the unimodal setting only the structure is sampled. We see that **including a jump model results in state-of-the-art performance while greatly increases the diversity metric**. See App. G.4 for more results and experiment details.

Method	Multimodal Div.	Multimodal Nov.	Unimodal Div.	Unimodal Nov.
RFdiffusion (Watson et al., 2023)	N/A		0.4	0.37
FrameFlow (Yim et al., 2024)	N/A		0.39	0.39
FoldFlow (Bose et al., 2023)	N/A		0.24	0.32
Protpardelle (Chu et al., 2024)	0.1	0.4	0.12	**0.41**
ProteinGenerator (Lisanza et al., 2023)	0.09	0.31	0.19	0.35
MultiFlow (Campbell et al., 2024b)	0.38	0.39	0.52	0.39
w/ $SO(3)$ jumps (ours)	**0.48**	**0.41**	**0.63**	**0.41**
w/ $SO(3)$ jumps + flow (ours)	0.47	0.4	0.59	0.40

Table 3: Protein generation results. Diversity (Div) is the share of *unique* proteins passing a quality check called designability, Novelty (Nov) is the average inverse similarity of each protein passing designability.

Additional small-scale experiments. We include a systematic study of the design space of GM on toy data in app. E. In app. I, we show that new Bregman divergences can improve existing flow and diffusion models.

10 Discussion

We introduced Generator Matching, a general framework for scalable generative modeling on arbitrary state spaces via Markov generators. The generator abstraction offers key insights into the fundamental equations governing Markov generative models: Generators are *linear* operators, the KFE is a *linear* equation, and Bregman divergences are *linear* in the training target. Therefore, any minimization we do *conditionally* on a data point, implicitly minimizes the training target *marginalized* across a distribution of data points. These principles allowed us to unify a diversity of prior generative modeling methods such as diffusion models, flow matching, or discrete diffusion models. Further, we could universally characterize the space of Markov models and loss functions. GM allows us to combine generative models of different classes on the same state space (Markov superpositions) and to easily build multimodal generative models. Future work could further explore the design space of GM. For example, we showed how one can learn a diffusion coefficient σ_t of a diffusion model. In addition, jump models on Euclidean space offer a large class of models that we could only study here in its simple instances. In addition, future work can explore better samplers or distillation. To conclude, Generator Matching provides both a rigorous theoretical foundation and opens up a large practical design space to advance generative modeling across a diverse range of applications.