The Flow of Creation: A Tour of Flow Matching for Visuals

Manolo Canales Cuba
Universidade Federal do ABC
Santo André, Brazil, 09280-560
Email: manolo.canales@ufabc.edu.br

Vinícius do Carmo Melício
Universidade Federal do ABC
Santo André, Brazil, 09280-560
Email: vinicius.melicio@ufabc.edu.br

João Paulo Gois
Universidade Federal do ABC
Santo André, Brazil, 09280-560
Email: joao.gois@ufabc.edu.br

***Abstract*—Flow Matching has recently emerged as an efficient alternative to the generative method paradigms. Here we aim to provide researchers and practitioners with both the theoretical and the practical aspects of this technique. We delve into the continuous formulation, where a neural network learns a vector field to transform noise into data via an ordinary differential equation, and also explore its discrete counterpart. The paper covers the entire workflow, from the core mathematical concepts and training objectives to sampling procedures, including classifier-free guidance and conditioning generation. By showcasing a diverse range of applications—from image synthesis and human motion generation to computational biology and robotics—this work equips readers with the essential knowledge to apply and innovate with the versatile and computationally efficient flow matching framework.**

***Index Terms*—Flow Matching, Discrete Generative Models, Deep Learning.**

I. Introduction

Generative modeling has become a cornerstone of modern artificial intelligence, yielding state-of-the-art results that are highly relevant to many fields, including computer graphics and vision. We can quickly exemplify some very popular generative models that have been used in various fields: text-to-image synthesis, such as DALL-E 3 and Stable Diffusion; large language models, including ChatGPT and Gemini; and video generation capabilities, like FLUX and Veo. One of the predominant approaches that has produced successes is the denoising diffusion model, which iteratively transforms a noise distribution into a sample from a complex data distribution, demonstrating effective performance across various data modalities, including images, audio, and 3D shapes.

Within this context of rapid innovation, Flow Matching emerges as a promising alternative to diffusion-based approaches for generative models, featuring faster inference times and a simplified mathematical formulation. By simulating an ordinary differential equation (ODE) with deep neural networks, Flow Matching presents a computationally efficient and conceptually elegant approach to generative modeling. Unlike denoising diffusion models, which rely on stochastic differential equations (SDE) and thus introduce stochasticity during inference, Flow Matching enables deterministic inference, contributing to a mathematically simpler process.

979-8-3315-8951-6/25/$31.00 ©2025 IEEE

This paper provides a practical and conceptual overview of Flow Matching, focusing on the continuous formulation introduced by Lipman et al. [1], with a dedicated section on its discrete counterpart [2]. The objective is to offer an accessible entry point for researchers and practitioners interested in understanding and implementing Flow Matching methods.

We present applications where Flow Matching has been successfully employed and conclude this document by highlighting some challenges, future directions, and additional topics not covered here.

II. Theoretical Foundations: Flow Matching

The theoretical foundation for Flow Matching, presented here, is drawn from the seminal contributions of Lipman et al. [1], [2], and the comprehensive materials of the online course provided by Holderrieth and Ezra [3].

The objective of Flow Matching is to transport points organized according to a simple and determined density p_0 to a more complex density p_1 that approximates the data distribution q. This transport of points is achieved through a function ψ_t. An initial point $x_0 = \psi_0(x_0)$ at time $t = 0$ is mapped to a final point $x_1 = \psi_1(x_0)$ at time $t = 1$ by the transport function ψ_t.

One approach to defining this function is to describe it using an ODE. To this end, we assume the existence of a vector field u_t:

$$\begin{aligned} \frac{d}{dt}\psi_t(x) &= u_t(\psi_t(x)), \\ \psi_0(x_0) &= x_0. \end{aligned} \tag{1}$$

Given this ODE, the existence and uniqueness of a solution ψ_t are guaranteed if certain regularity conditions are met by u_t [4]. These conditions ensure that the ODE is well-behaved, allowing for a stable and predictable solution. Because we will parameterize this vector field using a neural network, we implicitly satisfy these conditions, as neural networks typically have bounded derivatives [3]. The solution ψ_t then traces the path that carries x_0 to x_1 over time (Fig. 1). Consequently, ψ_t is a differentiable function with a differentiable inverse, called a *flow*. Similarly, u_t is time-dependent, and both are defined in the data space $\mathbb{R}^d$, where d is the dimensionality of the data.

At the same time, we want to ensure that the transport of these data maintains the existence of a density function p_t for all $t \in [0,1]$. We consider initial points $x_0 \sim p_0$ as elements of a random variable X_0. Our goal is to ensure that at any time t, the random variable X_t has a distribution p_t, thus $X_1 \sim p_1$. The set of functions $\{p_t\}_{t\in[0,1]}$ is called a *probability path* (Fig. 2). These time-dependent functions are also determined by the vector field u_t, which ensures their existence through a continuity equation:

$$\frac{d}{dt}p_t(x) + \operatorname{div}(p_t(x)u_t(x)) = 0. \tag{2}$$

When this condition is satisfied, we say that the vector field u_t generates p_t [1].

The vector field, flow, and probability path are thus related and dependent. It is worth noting that the ODE definition allows calculating the vector field from the flow, reinforcing the interconnectedness of these objects.

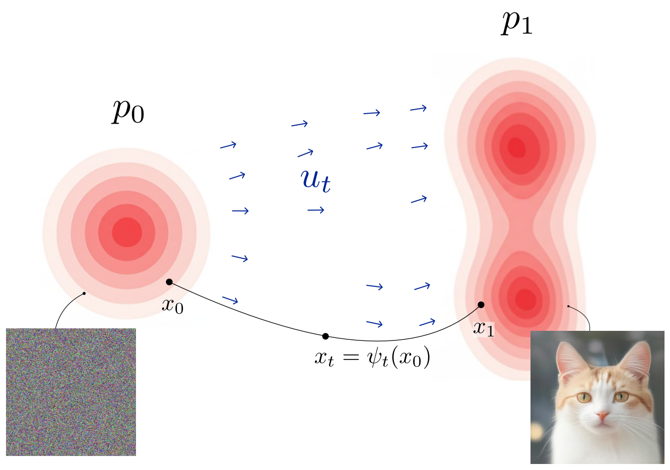

Fig. 1. Illustration of a solution trajectory ψ_t of the ODE defined by the vector field u_t. When the ODE is initialized with a point x_0, the solution ψ_t represents the trajectory of that point through the vector field. In this example, a noisy color image with distribution p_0 is transformed into a defined, noise-free final image with distribution p_1 at time $t = 1$.

Following the proposal of Chen et al. [5], we can parameterize the vector field u_t using deep neural networks and, in consequence, define the flow ψ and the probability path. We denote this parameterized vector field as u_t^θ, where $\theta \in \mathbb{R}^\ell$ represents its parameters. Then, the optimal choice of parameters θ can be obtained by minimizing the loss function:

$$\mathcal{L}_{\mathrm{FM}}(\theta) = \mathbb{E}_{t,x\sim p_t}\left\| u_t(x) - u_t^\theta(x) \right\|^2, \tag{3}$$

where $t \sim \mathcal{U}(0,1)$ is sampled from a uniform distribution.

A key challenge arises from the fact that the true vector field u_t is unknown. Furthermore, we only have an initial sample of points $\mathcal{D} = \{x_i\}_{i=1}^N$, $x_i \in \mathbb{R}^d$, from the data distribution q; that is, $x_i \sim q$. To address this challenge, Lipman et al. [1] propose constructing a vector field conditioned on $\mathcal{D}$, leveraging conditional probability paths and flows also constructed with respect to $\mathcal{D}$.

A. Construction of the Conditional Vector Field

Following Lipman et al. [1], a conditional probability path $p_t(\cdot|x_i)$ is constructed for each data point $x_i \in \mathcal{D}$. These

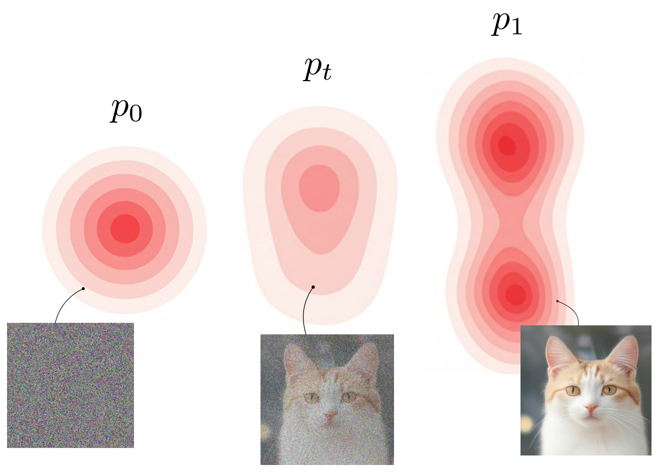

Fig. 2. Illustration of the probability path p_t showing the evolution of a noisy image (distribution p_0) through intermediate states to a clean image (distribution p_1).

paths are defined as Gaussian functions. At the initial time $t = 0$, as we know $p_0(\cdot|x_i) = p_0$, where p_0 is considered a normal distribution, specifically $p_0 = \mathcal{N}(0, \sigma^2 I)$, with $\sigma = 1$. Recalling that the goal of Flow Matching is to transform simple distributions into complex ones, the choice of a normal distribution as the initial distribution is coherent. Similarly, at the final time $t = 1$, we have $p_1(\cdot|x_i) = \mathcal{N}(x_i, \sigma_{min}^2 I)$, where the mean is x_i and the standard deviation $\sigma_{min} > 0$ is chosen to be sufficiently small to isolate x_i from other values in $\mathcal{D}$ (Fig. 3). Thus, $p_t(\cdot|x_i)$ is defined as the interpolation of means and standard deviations between these two extremes, yielding:

$$p_t(x|x_i) = \mathcal{N}(x; \mu_t(x_i), \sigma_t(x_i)^2 I), \tag{4}$$

where $\mu_t(x_i) = tx_i$ and $\sigma_t(x_i) = 1-(1-\sigma_{min})t$. Substituting into Eq. 4 yields:

$$p_t(x|x_i) = \mathcal{N}(x; tx_i, (1-(1-\sigma_{min})t)^2 I). \tag{5}$$

From this conditional probability path, a corresponding conditional flow $\psi_t(\cdot|x_i)$ can be constructed. This construction leverages the Gaussian probability path to define a function for transporting points. By employing the reparameterization trick, which allows transforming samples from one Gaussian distribution to another, we indirectly define this transport function. Specifically, given a point x_0 sampled from a normal distribution $x_0 \sim \mathcal{N}(0, I) = p_0$, we obtain the parameters (μ_t, σ_t) of the target distribution p_t. The point $\sigma_t(x_i)x_0 + \mu_t(x_i)$ then constitutes a sample from the distribution p_t. Thus, for some x_0, the conditional flow is generally defined as:

$$\psi_t(x_0|x_i) = \sigma_t(x_i)x_0 + \mu_t(x_i). \tag{6}$$

Substituting the corresponding expressions for σ_t and μ_t, we have:

$$\psi_t(x_0|x_i) = (1-(1-\sigma_{min})t)x_0 + tx_i. \tag{7}$$

This function has distribution p_t, is linear, and therefore differentiable with a differentiable inverse, meaning it is a diffeomorphism.

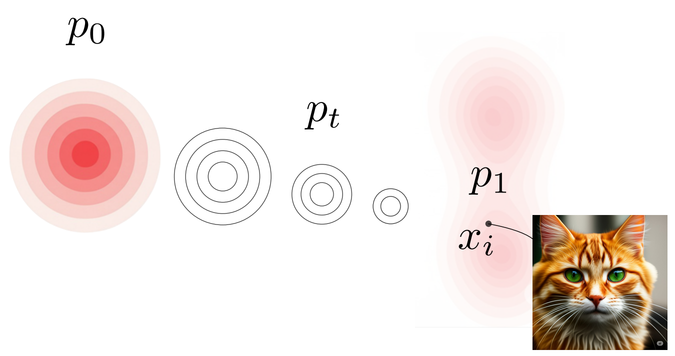

Fig. 3. Flow Matching as a transport from a normal distribution p_0 to a narrow Gaussian target p_1 centered at a dataset sample x_i (illustrated here by the cat). The intermediate distributions p_t describe a conditional probability path connecting p_0 and p_1.

For each $x_i \in \mathcal{D}$, we can define the conditional vector field $u_t(\cdot|x_i)$. Given the diffeomorphism $\psi_t(\cdot|x_i)$, we define an ODE, similar to Eq. 1, under which we can define $u_t(\cdot|x_i)$. Given that $x_t = \psi_t(x_0|x_i)$, then:

$$\begin{aligned} u_t(x_t|x_i) &= \frac{\partial}{\partial t}\psi_t(x_0|x_i) \\ &= \frac{\partial}{\partial t}\Big[\sigma_t(x_i)x_0 + \mu_t(x_i)\Big] \\ &= \frac{\partial}{\partial t}\Big[(1-(1-\sigma_{min})t)x_0 + tx_i\Big] \\ &= -(1-\sigma_{min})x_0 + x_i. \end{aligned}$$

Under this definition of u_t, we can define a parameterized deep neural network u_t^θ, such that the optimal parameter θ is found by minimizing the loss function:

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t, x_t\sim p_t(\cdot|x_i), x_i\sim q}\Big\|u_t(x_t|x_i) - u_t^\theta(x_t)\Big\|^2. \quad (8)$$

This function is called the Conditional Flow Matching (CFM) objective. It can be shown that the gradients of this objective are equal to those of the loss function in Eq. 3, as demonstrated in [1]. Thus, training the neural network under the objective in Eq. 8 allows us to find the network that approximates the main vector field (Eq. 3).

B. Model Training

In this section, we detail the training procedure for our model. The training process involves sampling data points from the dataset and learning to estimate the vector field that guides the generative flow. A pseudocode representation of the training algorithm is provided below (Alg. 1) for clarity.

Algorithm 1 Conditional Flow Matching Training Loop

```
1: for epoch = 1 to $E$ do
2:     for batch = 1 to $\lceil|\mathcal{D}|/B\rceil$ do
3:         Sample $x \in \mathbb{R}^{B\times d}$ from $\mathcal{D}$
4:         Sample $x_0 \in \mathbb{R}^{B\times d}$ from $\mathcal{N}(0, I)$
5:         Sample $t \in \mathbb{R}^B$ from Uniform$(0,1)$
6:         Compute $x_t = (1-(1-\sigma_{\min})\cdot t)\odot x_0 + t\odot x$ ▷ Eq. 7
7:         Evaluate $u_t^\theta(x_t)$ using the model
8:         Compute loss $\mathcal{L}_{\text{CFM}}$                ▷ Eq. 8
9:         Compute gradients $\nabla_\theta\mathcal{L}_{\text{CFM}}$
10:        Update parameters: $\theta \leftarrow \theta - \alpha\nabla_\theta\mathcal{L}_{\text{CFM}}$
11:    end for
12: end for
```

As is standard, the model is trained over a fixed number of epochs E with a predefined batch size B. During each iteration, we sample x from the dataset $\mathcal{D}$. For each x, a corresponding x_0 is sampled from the initial noise distribution $\mathcal{N}(0, I)$. Subsequently, a random time point $t \in [0,1]$ is selected. The point x_t is then determined as a linear interpolation between x_0 and x, according to Eq. 7, where $\odot$ represents the Hadamard product (element-wise multiplication). This time parameter represents the noise level present in x_t: when $t = 0$, x_0 represents a sample from the noise distribution, while when $t = 1$, x represents a sample from the data distribution without noise. Given x_t and the time t, we evaluate the vector field $u_t^\theta(x_t)$ using the previously defined model. The model parameters, θ, are then updated via gradient descent to minimize the loss function $\mathcal{L}_{CFM}$ (Eq. 8).

C. Sampling and Generation

This section outlines the procedure for generating new samples using the trained model. The generation process involves starting from a noise sample and iteratively refining it by following the learned vector field. To determine the flow of this vector field, we solve the ordinary differential equation (ODE) that relates them using the Euler method. This yields the iterative update:

$$x_{i+1} = x_i + hu_t^\theta(x_i), \quad (9)$$

where h is the step size. We present a pseudocode representation of the sampling algorithm for clarity.

Algorithm 2 Sampling Algorithm using Euler's Method

```
Require: Trained model $u_t^\theta$, Number of steps $n$, Step size $h = 1/n$
1: Sample $x_0 \sim \mathcal{N}(0, I)$
2: for $i = 0$ to $n-1$ do
3:     $t = i \cdot h$
4:     Evaluate $u_t^\theta(x_i)$ using the model
5:     Update $x_{i+1} = x_i + hu_t^\theta(x_i)$        ▷ Eq. 9
6: end for
7: return $x_n$
```

The generation process begins by sampling $x_0 \sim \mathcal{N}(0, I)$. This initial noise sample is then iteratively refined by following the learned vector field, according to Eq. 9, to produce a final sample.

D. Classifier-free Guidance

In many applications, it is desirable to guide the generation process using labels or conditioning information. Let us denote the data points as (x_i, y), where x_i is a data point and y represents a conditioning label. The label y can be a text embedding, a discrete class, or another type of conditioning information.

To incorporate this conditioning information, we can train the neural network to approximate the conditional vector field $u_t(x|y)$. The corresponding loss function is defined as:

$$\mathcal{L}_{\mathrm{CFM}}(\theta) = \mathbb{E}_{t, x_t \sim p_t, y \sim p(y)} \left\| u_t(x_t|y) - u_t^\theta(x_t) \right\|^2, \qquad (10)$$

where x_t is sampled from the conditional probability path $p_t(x|x_i)$ as before, and $u_t^\theta(x_t)$ now represents the neural network conditioned on the label y.

Following Ho and Salimans [6], we employ classifier-free guidance to improve the quality of the generated samples. The classifier-free guidance is defined as:

$$\tilde{u}_t(x|y) = (1-w)u_t(x|\emptyset) + w u_t(x|y). \qquad (11)$$

Here, $u_t(x|\emptyset)$ represents the unconditioned vector field, $u_t(x|y)$ represents the vector field conditioned on label y, and w is a guidance scale. The guidance scale controls the strength of the conditioning signal.

Critically, classifier-free guidance does not require training two separate models. Instead, the same neural network is trained to approximate both the conditional and unconditioned vector fields. During training, a fraction of the labels y are randomly replaced with a null symbol (e.g., $\emptyset$). This forces the network to learn to approximate the unconditioned vector field $u_t(x|\emptyset)$ when presented with this null symbol. In effect, a single network learns to handle both conditional and unconditional generation.

When $w = 0$, the generation is entirely unconditioned, relying solely on the prior distribution. As w increases, the generation becomes more strongly influenced by the conditioning label y.

III. Discrete Flow Matching

A. Motivation and Background

While the previous formulation of Flow Matching focused on continuous data, its principles can be extended to discrete domains such as text, structured graphs, or categorical data. Discrete Flow Matching (DFM) adapts the core idea of transforming a simple initial distribution into a complex data distribution but replaces the continuous-time ODE with a continuous-time Markov chain (CTMC) [2], [7]. This allows the model to learn a smooth, reversible path between a random state (e.g., a sequence of masked tokens) and a structured, coherent data sample. This extension opens up new avenues for non-autoregressive generative modeling in domains where continuous representations are not natural.

B. Training in the Discrete Setting

The training process for DFM is typically framed as learning a probability denoiser [7]. The model is trained to predict the Probability Mass Function (PMF) of the final, clean data state, $p_{1|t}(X_1|X_t)$, given a corrupted state X_t from an intermediate time t in a predefined probability path. This path, which connects a simple source distribution (e.g., masked tokens) to the target data distribution, is often constructed as a mixture of the source and target states [2]. The learning task is a straightforward classification objective optimized with a cross-entropy loss, where for each token i in a sequence, the model minimizes the cross-entropy loss:

$$\mathcal{L}_{\mathrm{CDFM}}(\theta) = -\mathbb{E}_{t,(X_0,X_1),X_t} \log p_{1|t}^{\theta,i}(X_1^i|X_t) + \mathrm{const.} \qquad (12)$$

While the ultimate goal is to generate the correct transition rates for the underlying Markov chain, these rates are derived directly from the learned denoiser PMF during sampling, making the training process itself stable and direct.

C. Discrete Sampling and Applications

Sampling from a trained discrete flow model is a generative process that reverses the training path, starting from a random discrete sequence, X_0, and iteratively refining it into a structured output [2]. At each time step t, the model uses the learned denoiser $p_{1|t}^\theta$ to calculate the probability velocity u_t, which defines the transition rates. For a simple mixture path defined by a scheduler κ_t, this velocity is given by [7]:

$$u_t^i(x^i|z) = \frac{1}{1-\kappa_t}\frac{d\kappa_t}{dt}[p_{1|t}^\theta(x^i|z) - \delta_{z^i}(x^i)], \qquad (13)$$

where z is the current sequence, x^i is a possible next token, and δ is the delta function. This velocity is then used to define the transition probability for updating each token in the sequence in parallel. For a small time step h, the probability of a token X_t^i transitioning to a new state y^i is:

$$\mathbb{P}(X_{t+h}^i = y^i|X_t) = \delta_{X_t^i}(y^i) + h \cdot u_t^i(y^i, X_t). \qquad (14)$$

The full sampling process, illustrated for a two-token sequence in Fig. 4, is detailed in Algorithm 3. This non-autoregressive technique has shown strong performance on a variety of tasks, including high-quality text generation, code generation, and even image generation, where pixel color values are treated as discrete tokens [7].

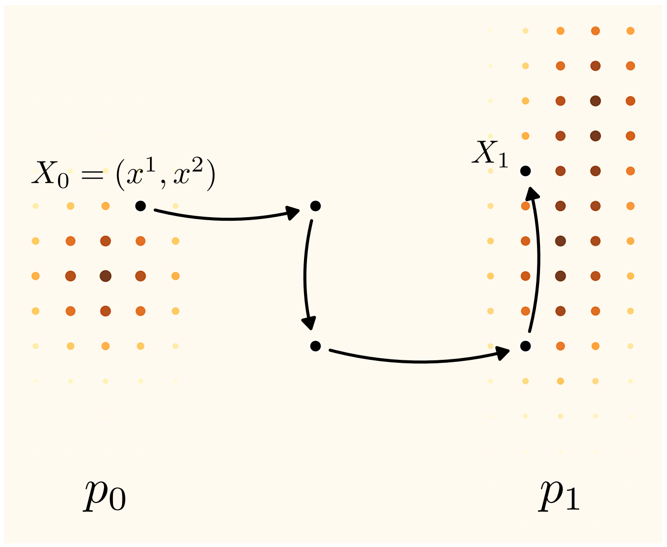

Fig. 4. Illustration of a factorized sampling path in a 2D discrete state space. The process starts at a state $X_0 = (x^1, x^2)$ in the source distribution p_0 and follows a series of horizontal and vertical jumps (changing one token at a time) to reach a final state X_1 in the target distribution p_1.

Algorithm 3 Discrete Flow Matching Sampling Loop

```
Require: Trained denoiser model $p^\theta_{1|t}$, number of steps $n$, step
    size $h = 1/n$.
1: Sample initial sequence $X_0$ from source distribution $p_0$.
2: for $k = 0$ to $n - 1$ do
3:     $t \leftarrow k/n$
4:     Evaluate denoiser $p^\theta_{1|t}(\cdot|X_k)$ to get clean data PMF.
5:     Calculate velocity $u_t$ for all tokens.            ▷ Eq. 13
6:     Sample next state $X_{k+1}$ from transition probabilities.
    ▷ Eq. 14
7: end for
8: return $X_n$
```

IV. Applications and Examples

Flow matching, due to its theoretical elegance and practical advantages, has found utility in diverse applications. This section highlights some prominent examples where flow matching has proven to be a valuable tool.

A. Image and Audio Generation

Recent works frequently employ various image generation algorithms using diffusion models and, more recently, conditional flow matching (CFM). Lipman et al. [1] demonstrated that training a CFM on the CIFAR-10 and ImageNet image datasets at 32, 64, and 128 resolution, without conditioning, achieves superior image generation results. They used similar architectures to alternative diffusion models such as DDPM, Score Matching (SM), and Score Flow (SF), showing improvements in fidelity (Fréchet Inception Distance, FID) compared to these methods.

Similarly, MimicTalk [8] demonstrates that videos of human faces can be generated from audio using CFM. This method then generates facial features guided by this audio, which are then rendered using pre-trained NeRF networks with LoRA. Likewise, the generation of the face can be coupled with the avatar's speech, as done in AV-Flow [9], which learns to generate audio and 4D video, interrelating them during generation using a diffusion transformer (DiT). From text, images, or audio of a user, a talking avatar is generated that responds accordingly in a conversational manner. They utilize CFM for the generation of the audio, the position of the avatar's head, and its facial features. Thanks to the inference speed of CFM, AV-Flow achieves faster performance than other similar methods.

B. Human Motion Generation

A promising application area is the generation of text-driven human motion. Models aim to create realistic and diverse human movements based only on textual descriptions. Flow matching has shown promising results in this area [10], offering a balance between generation speed and motion quality. A key challenge, however, is achieving temporally smooth motions, which is critical for creating conceivable animations. Recent work, such as FlowMotion [11], addresses this by incorporating target-predictive objectives within the CFM framework to directly encourage smoother and more stable motion sequences, even in resource-constrained environments.

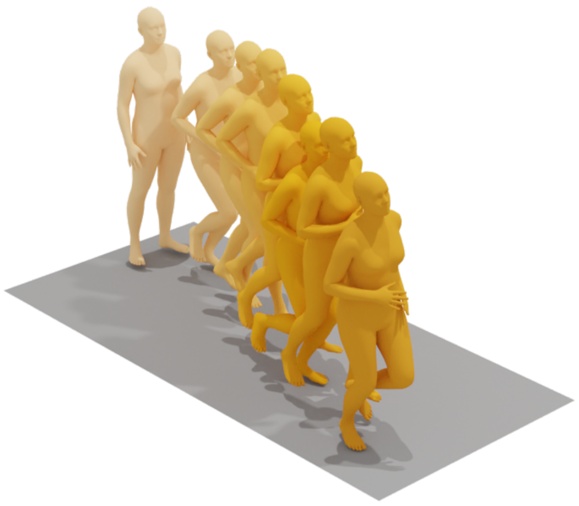

Fig. 5. Rendered snapshots of 8 frames from a motion sequence generated by FlowMotion. The prompt used was: "a person runs forward with arms bent at the elbow."

C. Computational Biology and Molecular Design

Flow matching is also being explored in computational biology, particularly for the complex task of designing novel proteins and molecules. The framework can learn to generate three-dimensional molecular structures that satisfy the strict geometric and chemical constraints required for biological function. This makes it a promising tool for applications such as creating new protein binders [12] or designing drug-like molecules (ligands) that fit into the specific binding pockets of target proteins [13]. By generating viable molecular candidates more efficiently, this approach has the potential to accelerate the therapeutic design and drug discovery pipeline significantly.

D. Robotics and Control Policies

In robotics, Flow Matching is emerging as a powerful method for learning robust control policies from demonstrations. The framework is used to generate smooth, continuous action trajectories and has been shown to be a more stable and computationally efficient alternative to diffusion-based policies, especially when requiring faster inference with fewer sampling steps [14]. Furthermore, Flow Matching can be integrated with specialized, geometry-aware network architectures to explicitly model the spatial relationships between a robot, its environment, and its actions [15]. This produces policies that are naturally equivariant, meaning they can better generalize to new object positions and orientations without needing to see every possible scenario during training, thus improving sample efficiency [15]. This combination of efficient trajectory generation and strong spatial reasoning makes Flow Matching a promising tool for developing more versatile and generalizable robots.

V. Conclusion and Future Directions

Flow Matching has emerged as a compelling alternative to existing generative modeling paradigms, offering a blend of theoretical rigor and practical efficiency. While diffusion models have recently gained prominence in the generative landscape, Flow Matching presents a competitive approach, particularly in scenarios where inference speed is essential. The ability to directly learn a vector field that maps noise to data, coupled with efficient ODE solvers, results in faster sampling times compared to iterative denoising processes inherent to diffusion models.

The applications discussed previously highlight the versatility of Flow Matching across diverse domains. However, certain trends and potential future directions warrant specific attention. The initial success of Flow Matching in areas like image generation is now extending to more complex modalities, such as video and 3D motion synthesis. In video generation, for instance, the relative speed advantage of Flow Matching is particularly significant, opening avenues for near-real-time video creation. Similarly, the efficient generation of realistic human motion sequences is proving valuable in animation and virtual reality applications.

Furthermore, the inherent efficiency of Flow Matching is especially advantageous in high-dimensional data domains, such as text generation and molecular design. Generating coherent and contextually relevant text requires processing vast amounts of information, making rapid inference a critical factor. The ability to rapidly create candidate molecules accelerates the drug discovery process.

Future research directions include exploring sophisticated network architectures for parameterizing the vector field, developing adaptive sampling schemes for ODE solvers to further accelerate inference, and investigating theoretical connections between Flow Matching and other generative frameworks. The development of novel loss functions that explicitly encourage properties like smoothness or disentanglement in generated samples also warrants investigation. Flow Matching will likely play an increasingly important role in generative modeling, pushing the boundaries of what is achievable in artificial intelligence and facilitating new applications across various scientific and engineering disciplines.

It is worth mentioning that many aspects were not covered in this document, for instance, the relation between Flow Matching and Diffusion Models [3] and manifold-based formulations [2], as well as the very recent novelties [16].

Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

References

[1] Y. Lipman, R. T. Chen, H. Ben-Hamu, M. Nickel, and M. Le, “Flow matching for generative modeling,” in *The Eleventh International Conference on Learning Representations*, 2022.

[2] Y. Lipman, M. Havasi, P. Holderrieth, N. Shaul, M. Le, B. Karrer, R. T. Chen, D. Lopez-Paz, H. Ben-Hamu, and I. Gat, “Flow matching guide and code,” *arXiv preprint arXiv:2412.06264*, 2024.

[3] P. Holderrieth and E. Erives, “Introduction to flow matching and diffusion models,” 2025. [Online]. Available: https://diffusion.csail.mit.edu/

[4] L. Perko, *Differential equations and dynamical systems*. Springer Science & Business Media, 2013, vol. 7.

[5] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, “Neural ordinary differential equations,” *CoRR*, vol. abs/1806.07366, 2018. [Online]. Available: http://arxiv.org/abs/1806.07366

[6] J. Ho and T. Salimans, “Classifier-free diffusion guidance,” *arXiv preprint arXiv:2207.12598*, 2022.

[7] I. Gat, T. Remez, N. Shaul, F. Kreuk, R. T. Chen, G. Synnaeve, Y. Adi, and Y. Lipman, “Discrete flow matching,” *Advances in Neural Information Processing Systems*, vol. 37, pp. 133 345–133 385, 2024.

[8] Z. Ye, T. Zhong, Y. Ren, J. Yang, W. Li, J. Huang, Z. Jiang, J. He, R. Huang, J. Liu, C. Zhang, X. Yin, Z. Ma, and Z. Zhao, “Mimictalk: Mimicking a personalized and expressive 3d talking face in few minutes,” 2024.

[9] A. Chatziagapi, L.-P. Morency, H. Gong, M. Zollhöfer, D. Samaras, and A. Richard, “Av-flow: Transforming text to audio-visual human-like interactions,” *arXiv preprint arXiv:2502.13133*, 2025.

[10] V. T. Hu, W. Yin, P. Ma, Y. Chen, B. Fernando, Y. M. Asano, E. Gavves, P. Mettes, B. Ommer, and C. G. Snoek, “Motion flow matching for human motion synthesis and editing,” *arXiv preprint arXiv:2312.08895*, 2023.

[11] M. Canales Cuba, V. do Carmo Melício, and J. P. Gois, “Flowmotion: Target-predictive conditional flow matching for jitter-reduced text-driven human motion generation,” *Computers & Graphics*, vol. 132, p. 104374, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0097849325002158

[12] J. Yan, Z. Cui, W. Yan, Y. Chen, M. Pu, S. Li, and S. Ye, “Robust and reliable de novo protein design: A flow-matching-based protein generative model achieves remarkably high success rates,” *bioRxiv*, pp. 2025–04, 2025.

[13] J. Cremer, R. Irwin, A. Tibo, J. P. Janet, S. Olsson, and D.-A. Clevert, “Flowr: Flow matching for structure-aware de novo, interaction-and fragment-based ligand generation,” *arXiv preprint arXiv:2504.10564*, 2025.

[14] F. Zhang and M. Gienger, “Affordance-based robot manipulation with flow matching,” *arXiv preprint arXiv:2409.01083*, 2024.

[15] N. Funk, J. Urain, J. Carvalho, V. Prasad, G. Chalvatzaki, and J. Peters, “Actionflow: Equivariant, accurate, and efficient policies with spatially symmetric flow matching,” *arXiv preprint arXiv:2409.04576*, 2024.

[16] N. Shaul, U. Singer, I. Gat, and Y. Lipman, “Transition matching: Scalable and flexible generative modeling,” 2025. [Online]. Available: https://arxiv.org/abs/2506.23589