Мультимодальные и поведенческие модели

Diffusion and Flow Matching

MIPT VLA 2026 | 14 сентября 2026

Алексей Староверов



МФТИ

Center for Cognitive Modeling

План лекции

From Generation to Sampling. Construct Flow and Diffusion Models

Dispelling the Myths of Generative Robotic Control

From Generation to Sampling. Construct Flow and Diffusion Models

We represent images/videos/protein as vectors

Images:

- Height H and Width W
- 3 color channels (RBG)

$$z \in \mathbb{R}^{H \times W \times 3}$$



Videos:

- T time frames
- Each frame is image

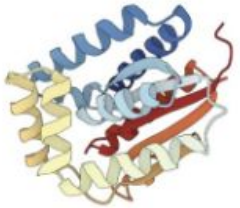
$$z \in \mathbb{R}^{T \times H \times W \times 3}$$



Molecular structures:

- N atoms
- Each atom has 3 coordinate

$$z \in \mathbb{R}^{N \times 3}$$

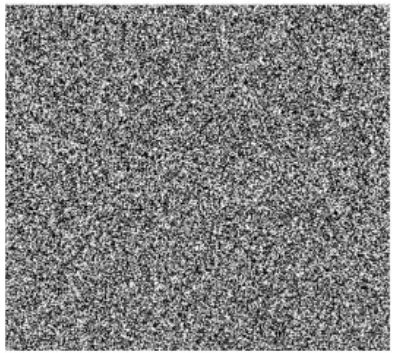


We represent the objects we want to generate as vectors:

$$z \in \mathbb{R}^{d}$$

What does it mean to successfully generate something?

Prompt: “A picture of a dog”



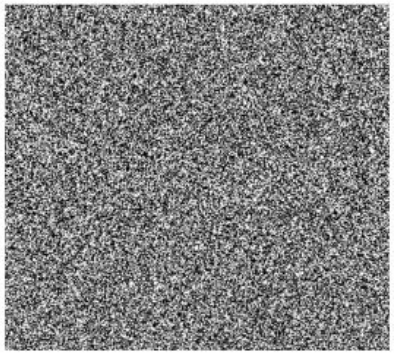



Useless < Bad < Wrong animal < Great!

These are subjective statements - Can we formalize this?

Data Distribution: How “likely” are we find this picture in the internet?

Prompt: “A picture of a dog”






Impossible < Rare < Unlikely < Very likely

How good an image is ~= How likely it is under the data distribution

Data Distribution: How "likely" are we find this picture in the internet?

Data distribution: Distribution of objects that we want to generate: p_{data}

Probability density:

$$p_{\text{data}} : \mathbb{R}^d \to \mathbb{R}_{\geq 0},$$

$$z \mapsto p_{\text{data}}(z)$$

Note: We don't know the probability density!

Generation means sampling the data distribution:

$$z \sim p_{\text{data}} \quad \Rightarrow \quad z =$$



A Dataset consists of samples from the data distribution

Training requires datasets: To train our algorithms, we need a **dataset**.

Examples:

- Images: Publicly available images from the internet
- Videos: YouTube
- Protein structures: Scientific data (e.g. Protein Data Bank)

A dataset consists of a finite number of samples from the data distribution:

$$z_1, \ldots, z_N \sim p_{\text{data}}$$

Conditional Generation allows us to condition on prompts

Data distribution p_{data}

Fixed prompt

"Dog"



Conditional data distribution $p_{\text{data}}(\cdot|y)$

Condition variable: **y**

y="Dog" y="Cat" y="Landscape"



Conditional generation means sampling the conditional data distribution:

$$z \sim p_{\text{data}}(\cdot|y)$$

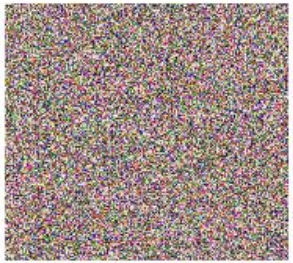
We will first focus on unconditional generation and then learn how to translate an unconditional model to a conditional one.

Generative Models generate samples from data distribution

Initial distribution: p_{init} ***Default:*** $p_{\text{init}} = \mathcal{N}(0, I_d)$

A generative model converts samples from a initial distribution (e.g. Gaussian) into samples from the data distribution:

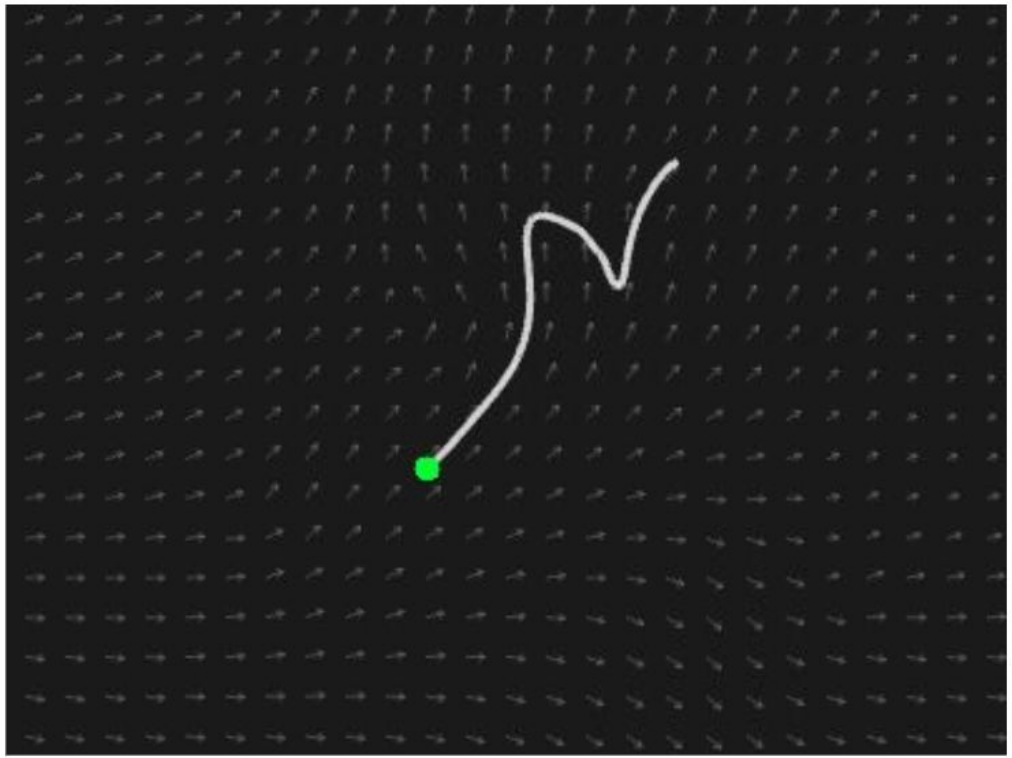
$x \sim p_{\text{init}}$ → Generative Model → $z \sim p_{\text{data}}$





Flow and Diffusion Models

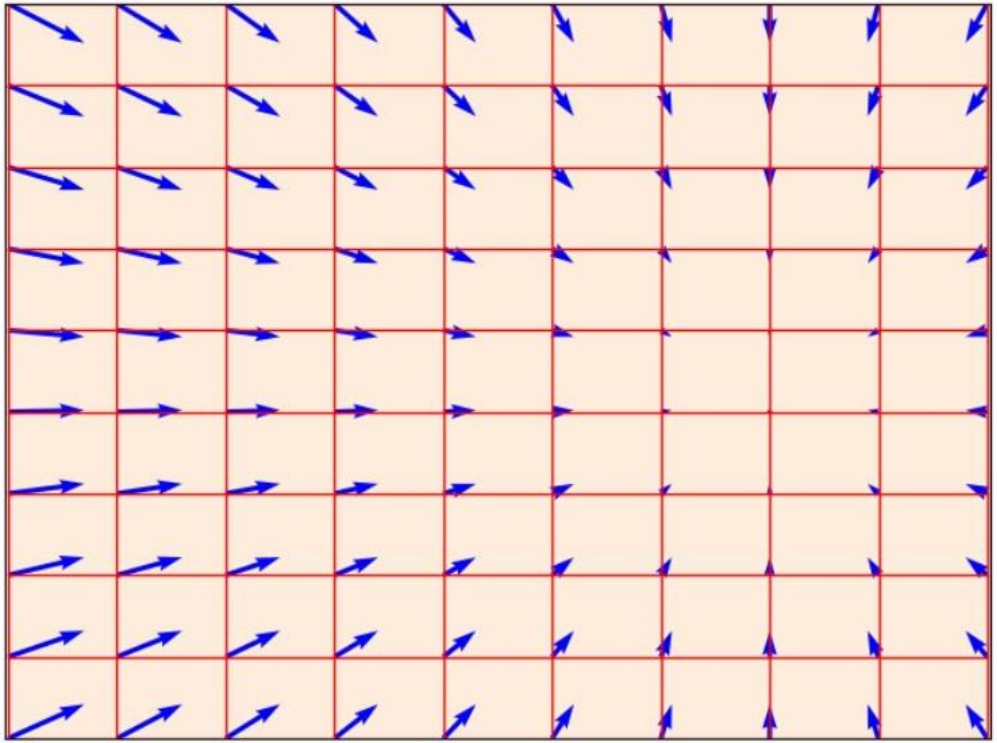
ODE Trajectory
- Example



Source: https://mariogemoll.com/flow-matching

Flow and Diffusion Models

Flow - Example



Flow and Diffusion Models

Theorem (Picard–Lindelöf theorem): If the vector field $u_t(x)$ is continuously differentiable with bounded derivatives, then a *unique* solution to the ODE

$$X_0 = x_0, \qquad \frac{\mathrm{d}}{\mathrm{d}t} X_t = u_t(X_t)$$

exists. In other words, a flow map exists. More generally, this is true if the vector field is **Lipschitz**.

Numerical ODE simulation - Euler method

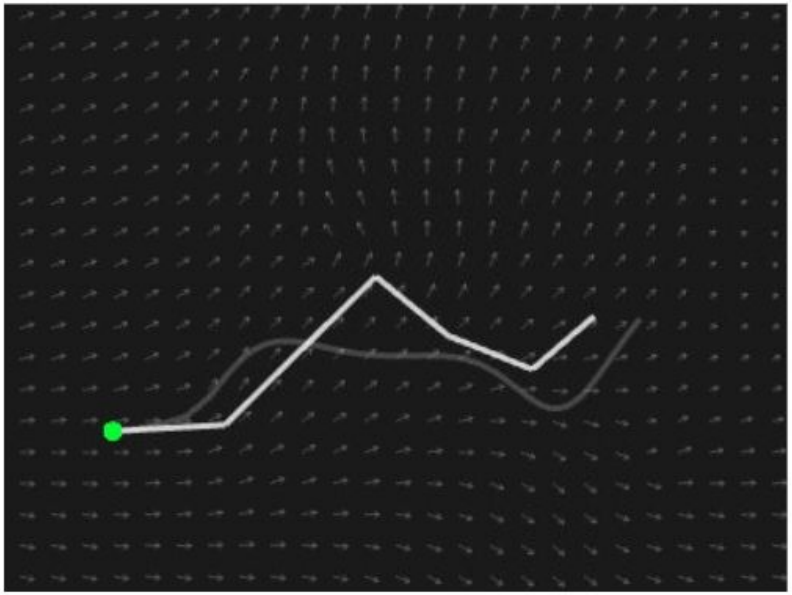
Algorithm 1 Simulating an ODE with the Euler method

Require: Vector field u_t, initial condition x_0, number of steps n

Set $t = 0$
Set step size $h = \frac{1}{n}$
Set $X_0 = x_0$
for $i = 1, \ldots, n-1$ **do**
    $X_{t+h} = X_t + hu_t(X_t)$ *Small step into direction of vector field*
    Update $t \leftarrow t + h$
end for
return $X_0, X_h, X_{2h}, \ldots, X_1$ *Return trajectory*

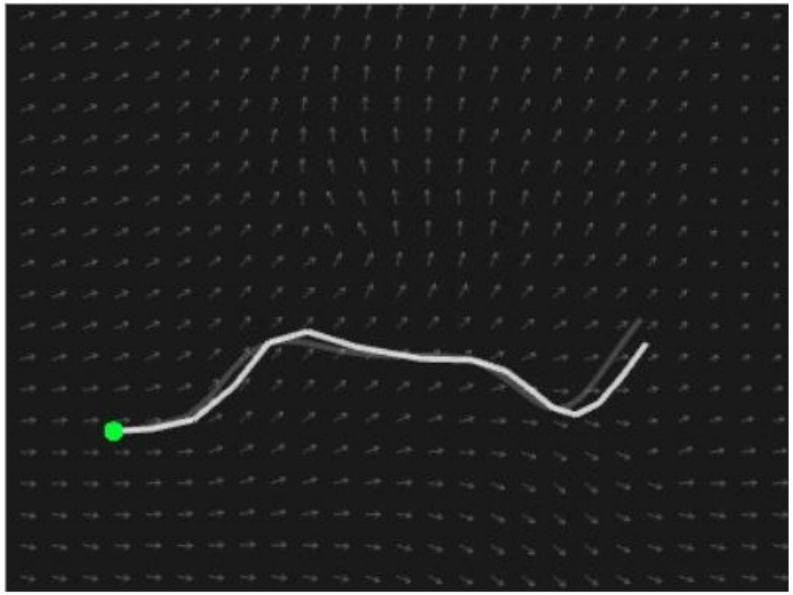
Euler Method - Example

Large step size



More efficient but *higher error*

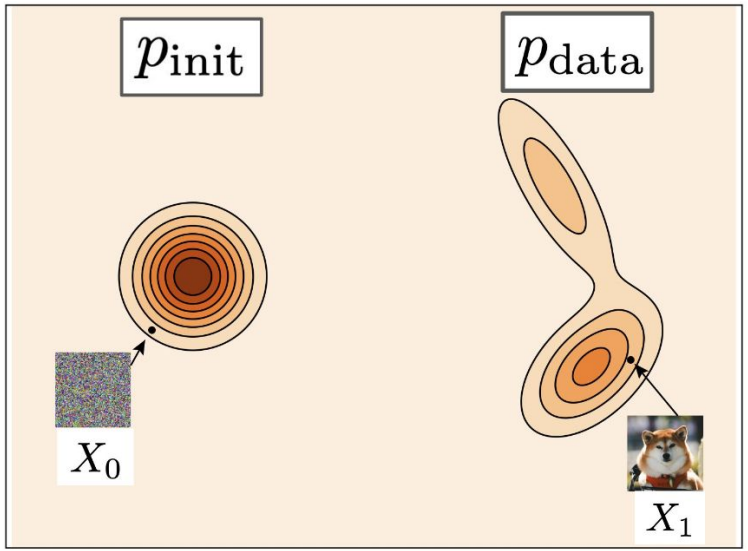
Small step size

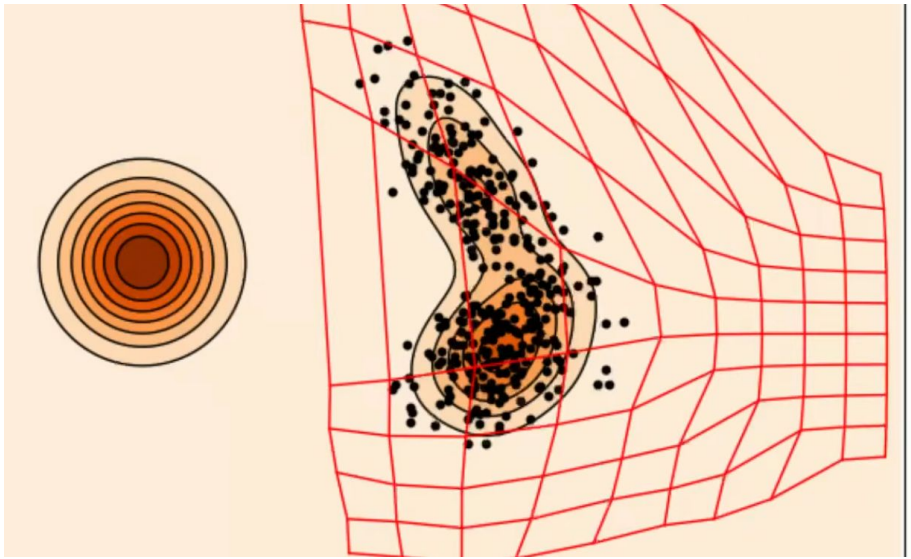


Lower error but *less efficient*

Source: https://mariogemoll.com/flow-matching

Euler Method - Example





Flow and Diffusion Models

Flow Model

Initialize: $X_0 \sim p_{\text{init}},$

ODE: $\mathrm{d}X_t = u_t^\theta(X_t)\mathrm{d}t$

E.g. Gaussian

Neural network vector field

Diffusion coeff.

Diffusion Model

Initialize: $X_0 \sim p_{\text{init}},$

SDE: $\mathrm{d}X_t = u_t^\theta(X_t)\mathrm{d}t + \sigma_t \mathrm{d}W_t$

To get samples, simulate ODE/SDE from t=0 to t=1 and return X_1

Training a Flow Model

Without training, the model produces “non-sense”→ **We need to train** u_t^θ

Training = Finding parameters θ **such that**

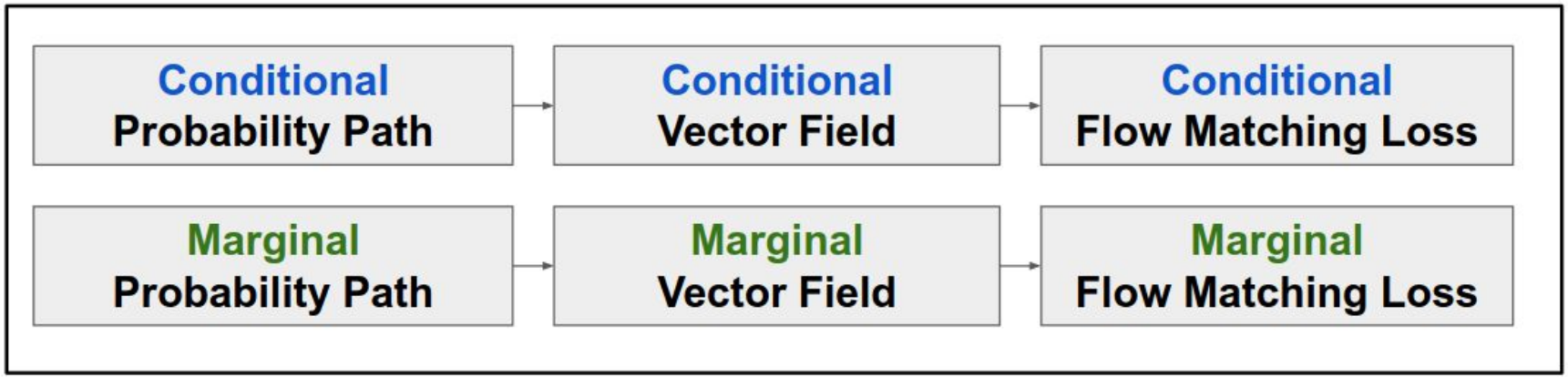
$$X_0 \sim p_{\text{init}}, \quad \mathrm{d}X_t = u_t^\theta(X_t)\mathrm{d}t \quad \Rightarrow \text{Implies} \quad X_1 \sim p_{\text{data}}$$

Start with initial distribution

Follow along the vector field

The distribution of the final point = data distribution

The Flow Matching Matrix

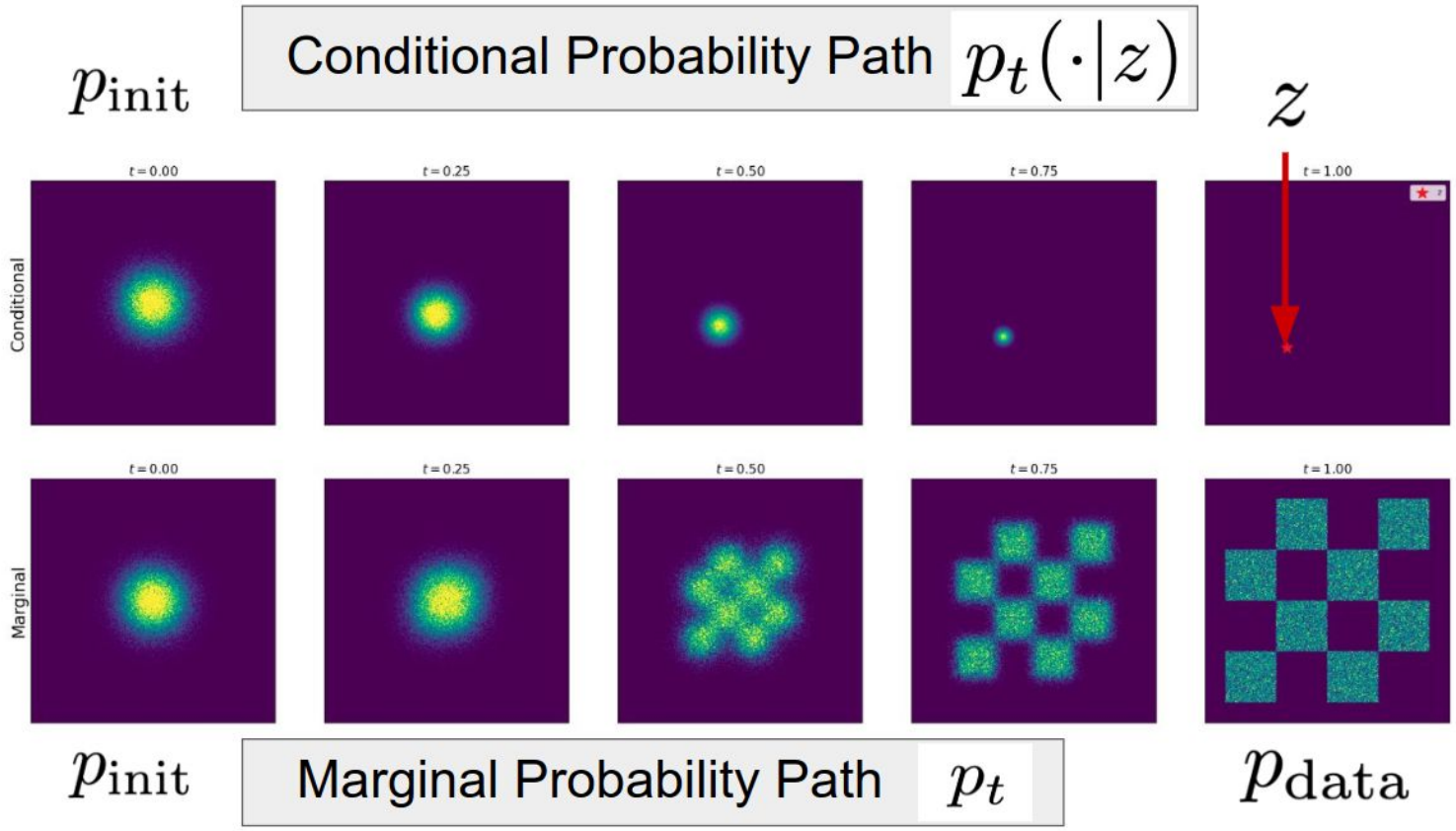


"Conditional" = "Per single data point"

"Marginal" = "Across distribution of data points"

The Flow Matching Matrix

Conditional Probability Path	**Conditional** Vector Field	**Conditional** Flow Matching Loss
Marginal Probability Path	**Marginal** Vector Field	**Marginal** Flow Matching Loss
Defines distributions from noise to data	Defines training target that we want to learn	Loss function that we want to minimize during traing


Conditional Probability Path $p_t(\cdot|z)$
p_{init}
z
$t = 0.00$
$t = 0.25$
$t = 0.50$
$t = 0.75$
$t = 1.00$
Conditional
Marginal
p_{init}
Marginal Probability Path p_t
p_{data}

Conditional Prob. Path, Vector Field, and FM Loss

	Notation	Key property	Gaussian example
Conditional Probability Path	$p_t(\cdot\|z)$	Interpolates p_{init} and a data point z	$\mathcal{N}(\alpha_t z, \beta_t^2 I_d)$
Conditional Vector Field	$u_t^{\text{target}}(x\|z)$	ODE follows conditional path	$\left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t}\alpha_t\right) z + \frac{\dot{\beta}_t}{\beta_t} x$
Conditional FM Loss	$L_{\text{CFM}}(\theta)$	Loss we minimize during training	$\mathbb{E}_{t,z,x}[\|\|u_t^\theta(x) - u_t^{\text{target}}(x\|z)\|\|^2]$

Marginal Prob. Path, Vector Field, and FM Loss

	Notation	Key property	Formula
Marginal Probability Path	p_t	Interpolates p_{init} and p_{data}	$\int p_t(x\|z)p_{\text{data}}(z)\mathrm{d}z$
Marginal Vector Field	$u_t^{\text{target}}(x)$	ODE follows marginal path	$\int u_t^{\text{target}}(x\|z)\frac{p_t(x\|z)p_{\text{data}}(z)}{p_t(x)}\mathrm{d}z$
Marginal FM Loss	$L_{\text{FM}}(\theta)$	Implicitly minimized via cond FM loss	$\mathbb{E}_{t,z,x}[\|\|u_t^\theta(x) - u_t^{\text{target}}(x)\|\|^2]$

None of these objects are tractable. But we can still learn them!

Что такое Flow Matching

Flow Matching обучает векторное поле, которое переводит шум в данные.

Генерация выполняется через решение обыкновенного дифференциального уравнения.

Начальная точка обычно сэмплируется из стандартного нормального распределения.

$$\frac{dx_t}{dt} = v_\theta(x_t, t)$$

$$x_0 \sim \mathcal{N}(0, I)$$

Вероятностный поток

Плотность распределения меняется во времени вместе с векторным полем.

Уравнение неразрывности выражает сохранение вероятностной массы.

Цель обучения — перевести начальное распределение в распределение данных.

$$\frac{\partial p_t}{\partial t} + \nabla \cdot (p_t v_t) = 0$$

Линейный путь между шумом и данными

Часто используется линейная интерполяция между шумом и данными.

Целевая скорость равна производной траектории по времени.

$$x_t = (1 - t)x_0 + tx_1$$

$$u_t = \frac{dx_t}{dt} = x_1 - x_0$$

Условная плотность для линейного пути

Для фиксированного целевого значения промежуточное распределение является гауссовым.
По мере роста времени распределение стягивается к данным.

$$p_t(x|x_1) = \mathcal{N}(x; tx_1, (1-t)^2 I)$$

$$u_t(x|x_1) = \frac{x_1 - x}{1 - t}$$

Идеальная цель Flow Matching

В теоретической постановке модель сопоставляется с истинным векторным полем.

На практике истинное маргинальное поле неизвестно.

$$\mathcal{L}_{FM}(\theta) = \mathbb{E}_{t\sim\mathcal{U}[0,1],x\sim p_t}\left[\|v_\theta(x,t) - u_t(x)\|_2^2\right]$$

Условный Flow Matching

Условный путь позволяет обучать модель без знания маргинального поля.

Для линейного пути задача становится регрессией на разность данных и шума.

$$\mathcal{L}_{CFM}(\theta) = \mathbb{E}_{t,x_0,x_1}\left[\|v_\theta((1-t)x_0 + tx_1, t) - (x_1 - x_0)\|_2^2\right]$$

Эквивалентность и оптимальное решение

Градиенты условной и маргинальной цели совпадают.

Оптимальная модель предсказывает условное среднее целевой скорости.

$$\nabla_\theta \mathcal{L}_{FM} = \nabla_\theta \mathcal{L}_{CFM}$$

$$v^*(x_t, t) = \mathbb{E}[x_1 - x_0 \mid x_t]$$

Постановка задачи для π0

π0 генерирует блок действий робота по наблюдениям, языковой инструкции и состоянию. Условие формируется мультимодальным кодировщиком.

$$c = \mathrm{Enc}(o, l, s)$$

$$a \in \mathbb{R}^{H \times d_a}$$

Основная формула обучения π0

Модель предсказывает скорость от шума к целевому блоку действий.
Обучение выполняется по демонстрациям с квадратичным лоссом.

$$\mathcal{L}_{\pi_0}(\theta) = \mathbb{E}_{\mathcal{D},\varepsilon,t}\left[\|v_\theta(a_t, t, c) - (a - \varepsilon)\|_F^2\right]$$

$$(o, l, s, a) \sim \mathcal{D}, \quad \varepsilon \sim \mathcal{N}(0, I), \quad t \sim \mathcal{U}[0, 1]$$

$$a_t = (1 - t)\varepsilon + ta, \quad c = \mathrm{Enc}(o, l, s)$$

Мини-батч версия

На практике минимизируется среднее значение ошибки по мини-батчу.

$$\mathcal{L}_{\pi_0}^{\text{batch}}(\theta) = \frac{1}{B}\sum_{i=1}^{B}\left\| v_\theta(a_{t_i}^{(i)}, t_i, c_i) - (a_i - \varepsilon_i)\right\|_F^2$$

Алгоритм обучения

1. Сэмплировать демонстрацию (o, l, s, a).
2. Сэмплировать шум ε и время t.
3. Построить зашумленное действие a_t.
4. Предсказать скорость и сравнить её с целью a - ε.

$$a_t = (1 - t)\varepsilon + ta$$

$$v_{\text{target}} = a - \varepsilon$$

Инференс π0

На этапе инференса модель интегрируется от шума к действию.
Число шагов интегрирования подбирается под качество и задержку управления.

$$a_0 \sim \mathcal{N}(0, I)$$

$$\frac{da_t}{dt} = v_\theta(a_t, t, c)$$

$$a_{k+1} = a_k + \Delta t \, v_\theta(a_k, t_k, c)$$

Итог

Flow Matching обучает векторное поле, переводящее шум в данные.

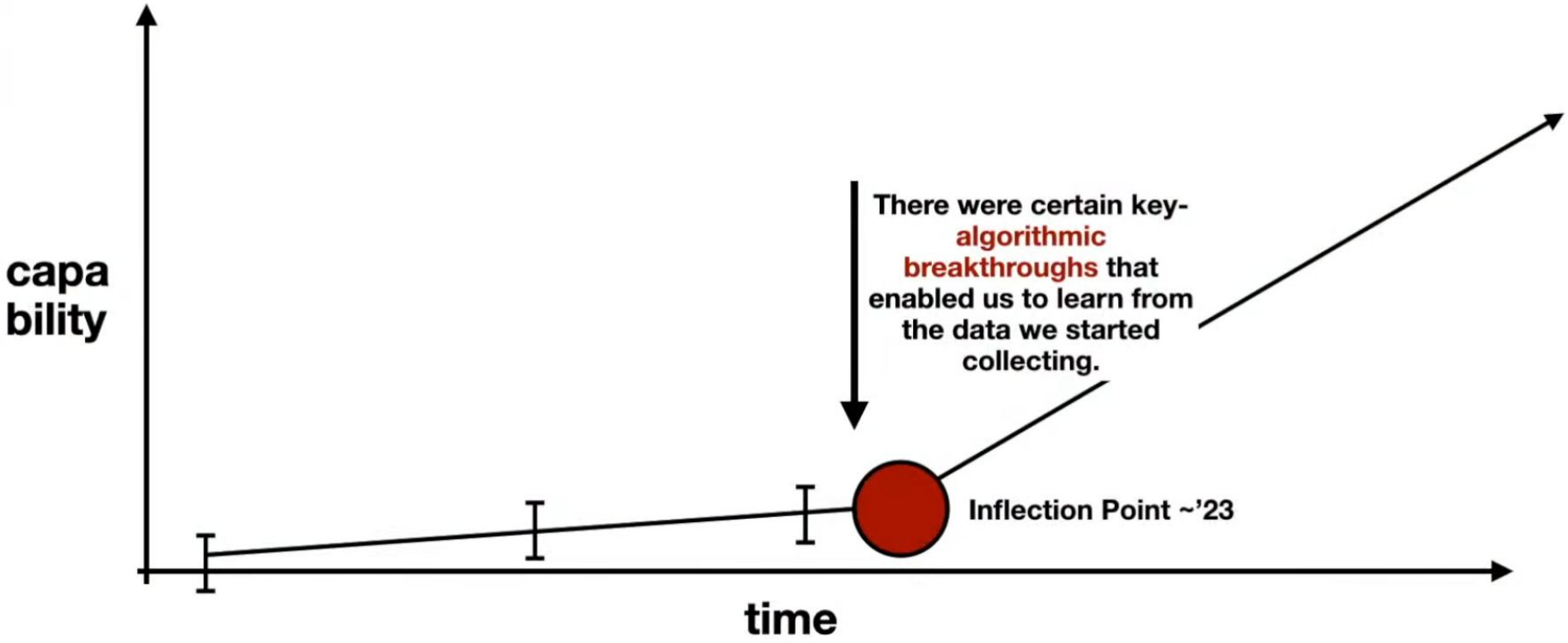
π0 обучается предсказывать скорость от шума к действию.

Главная формула обучения — регрессия скорости.

$$\mathcal{L}_{\pi_0}(\theta) = \mathbb{E}_{\mathcal{D},\varepsilon,t}\left[\|v_\theta(a_t, t, c) - (a - \varepsilon)\|_F^2\right]$$

Dispelling the Myths of Generative Robotic Control

Inflection point of Robot learning



What caused the inflection point?

Version 1: Getting data for robots is just harder

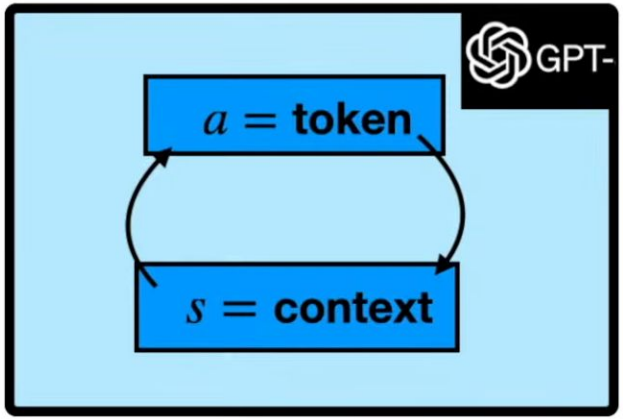
Version 2: Robot learning can be fundamentally harder without a few key algorithmic choices

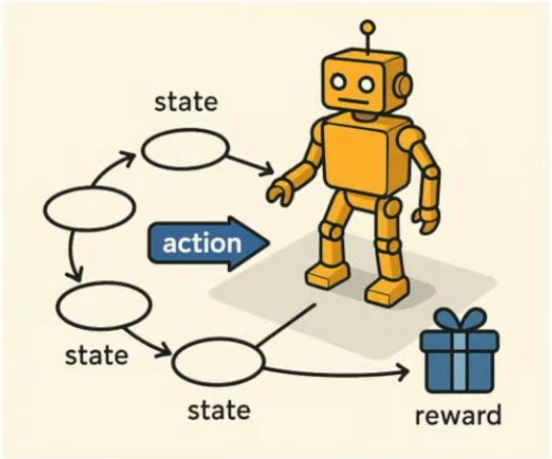
Today we will focus on one such inflection point: the use of **diffusion policies** (flows, interpolants, etc.) as **parameterizations of robot actions.**

(Widely considered the central innovation in robot learning)

Key Object: Policies

Definition: A **Policy** is a mapping $\pi : S \to \Delta(A)$



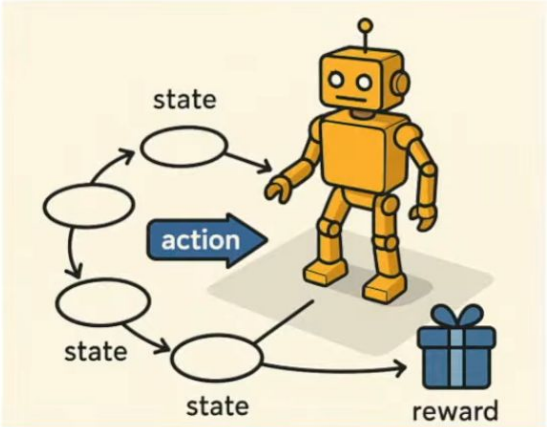


*Markovian policy

Desideratum: Expected Reward

Definition: The **J-functional** is $J(\pi) = \mathbb{E}^{\pi}\left[\sum_{t=1}^{H} r(s_t, a_t)\right]$

Expected cumulated reward of policy π



$$\mathbb{E}^{\pi} : s_{t+1} \sim P(s_t, a_t),\ a_t \sim \pi(s_t), s_1 \sim P_1$$

Protocol: Offline Imitation Learning

1. An **expert** $\pi^\star$ collects $n \in \mathbb{N}$ i.i.d trajectories $(s_1, a_1, \ldots, s_H, a_H) \sim \mathbb{P}^{\pi^\star}$

2. Given trajectories, **imitator** attempts to return $\hat{\pi}$ such that

$$J(\pi^\star) - J(\hat{\pi}) \leq \epsilon = o_n(1)$$

cumulative reward

Algorithm: Behavior Cloning

Behavior Cloning: Learner selects $\hat{\pi}$ to minimize **supervised learning error**

$$\hat{\pi} \approx \arg\min_{\pi \in \Pi} \sum_{i=1}^{n} \sum_{t=1}^{H} \text{loss}(s_t^{(i)}, a_t^{(i)}, \pi)$$

expert data

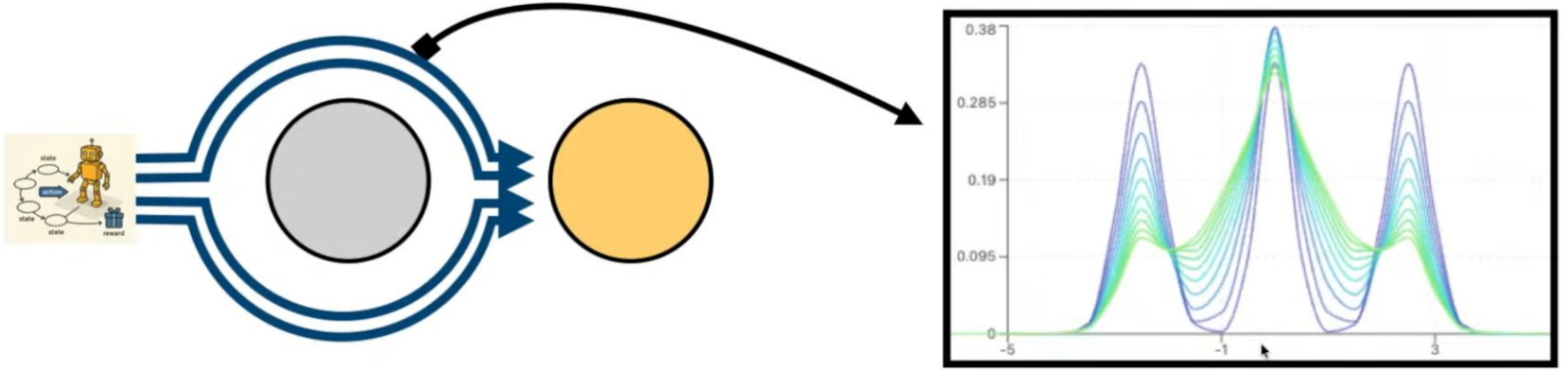
policy class

(e.g. neural network)

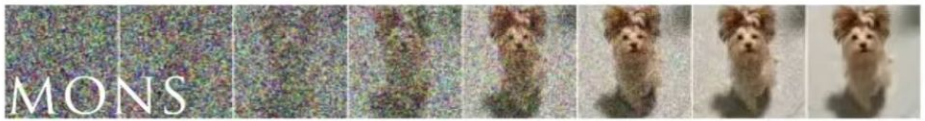
loss function

(related to policy parametrization)

Generative Control Policies (GCPs)



Train policy for predict **multimodal distributions of** $\pi^{\star}(a \mid o)$



(Diffusion Policy, Chi et al. '23)

Generative Control Policies (GCPs)

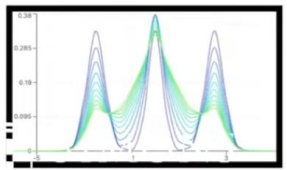
Behavior Cloning: Learner selects $\hat{\pi}$ to minimize **supervised learning error**

$$\hat{\pi} \approx \arg\min_{\pi \in \Pi} \sum_{i=1}^{n} \sum_{t=1}^{H} \text{loss}(s_t^{(i)}, a_t^{(i)}, \pi)$$

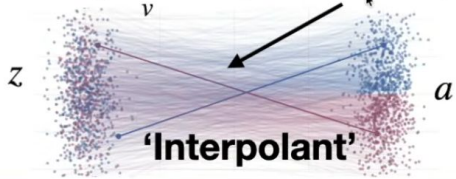
Traditional: Regression

$$\hat{\pi}_0 \approx \arg\min_{\pi} \mathbb{E}^{\pi^\star} \|\pi(s) - a\|^2$$

replace this with a distribution-fitting objective

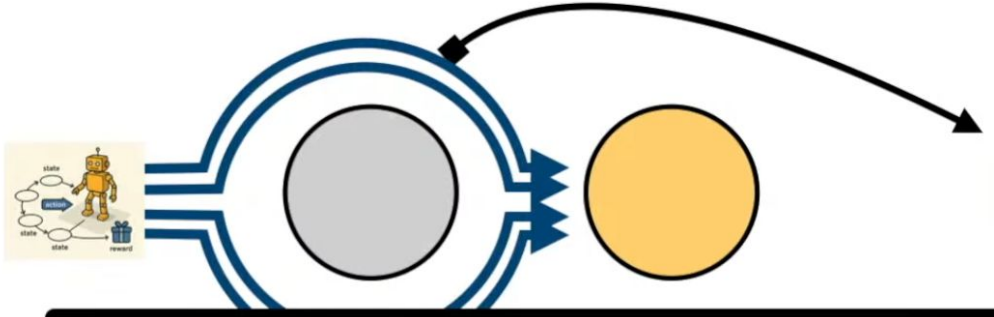


$$\hat{v} \approx \arg\min_{v} \mathbb{E}^{\pi^\star} \|v(ta + (1-t)z \mid t, o) - (z - a)\|^2,$$



'velocities'

Generative Control Policies (GCPs)



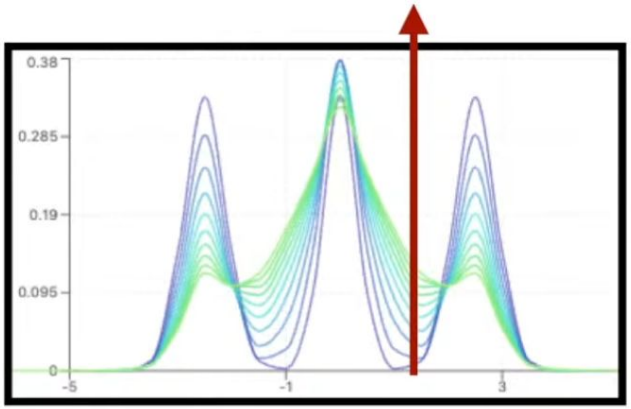
Train policy for predict **multimodal distributions of** $p(a \mid o)$

Again, to stress: generative control policies are the technology claimed to responsible for much of the recent **revolution in robot capabilities!**

Specifically, are GCPs really about multi-modal **distribution matching,** or are they are secretly doing … **something else**?

Generative Control Policies (GCPs)

Data modes are 'good' actions

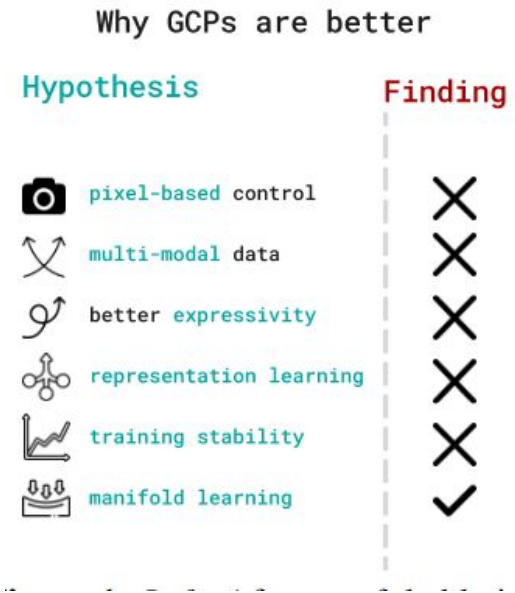


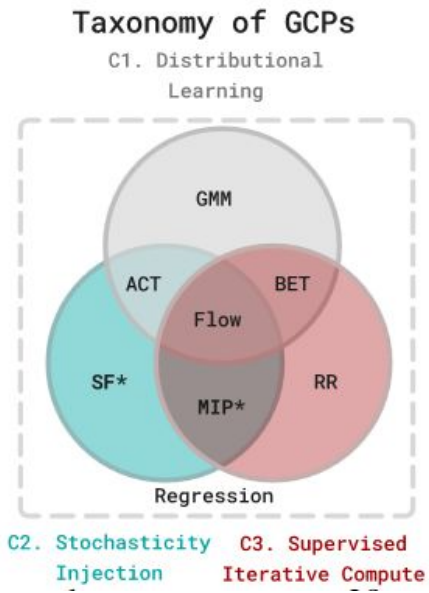
Yet still, it makes sense to use generative policies because **it is better to learn a mode than a mean…**

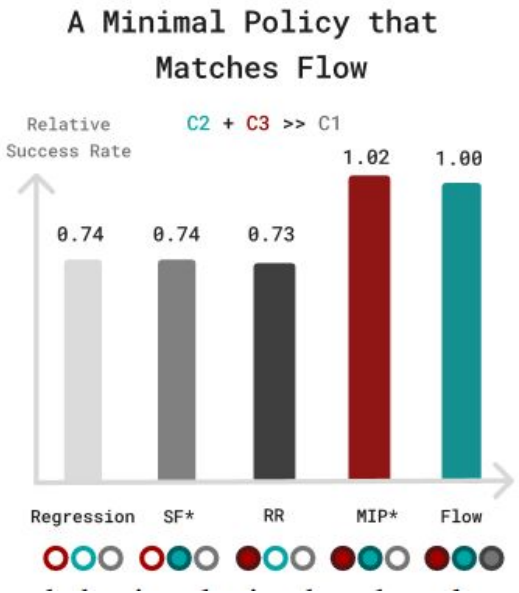
Recall: The **J-functional** is $J(\pi) = \mathbb{E}^{\pi}\left[\sum_{t=1}^{H} r(s_t, a_t)\right]$

Takeaway: unlike generative modeling, we just need one good action…

Why generative control policies (GCPs) outperform regression policies (RCP) on these tasks





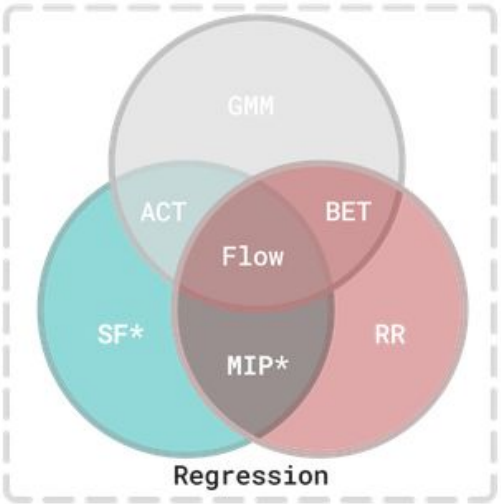


minimal iterative policy (MIP), straight-flow (SF) and residual regression (RR)

https://arxiv.org/abs/2512.01809

A Taxonomy of Control Policies (GCPs)

C1. Distributional Learning



C1. Distributional Learning: matching the exact conditional distribution of action given observation



C2. Stochasticity Injection: injecting noise during training to improve the learning dynamics.



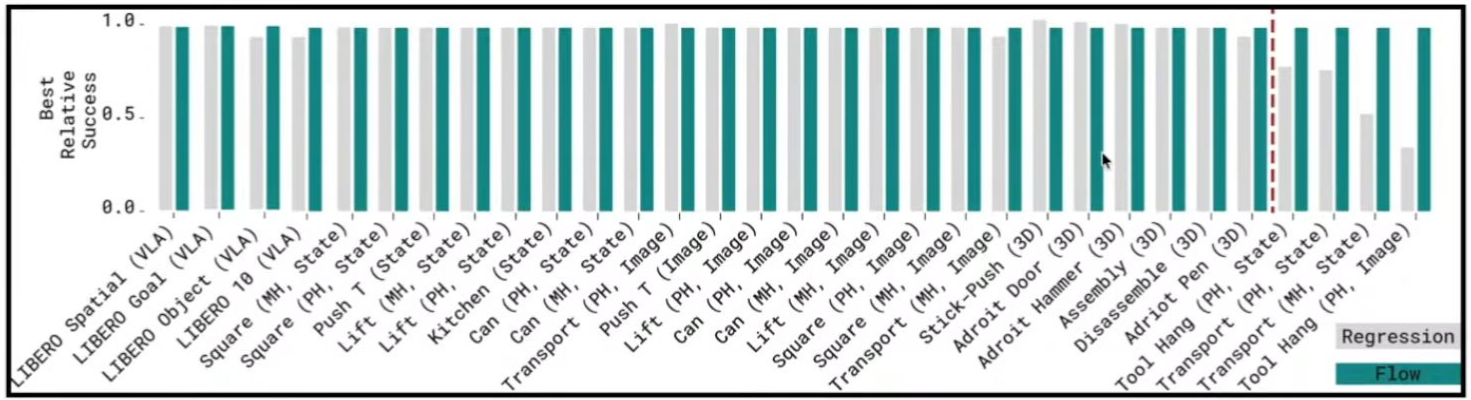
C3. Supervised Iterative Compute: generating output with multiple steps, each of which receives supervision during training



C2. Stochasticity Injection

C3. Supervised Iterative Compute

Are GCPs doing distribution learning?

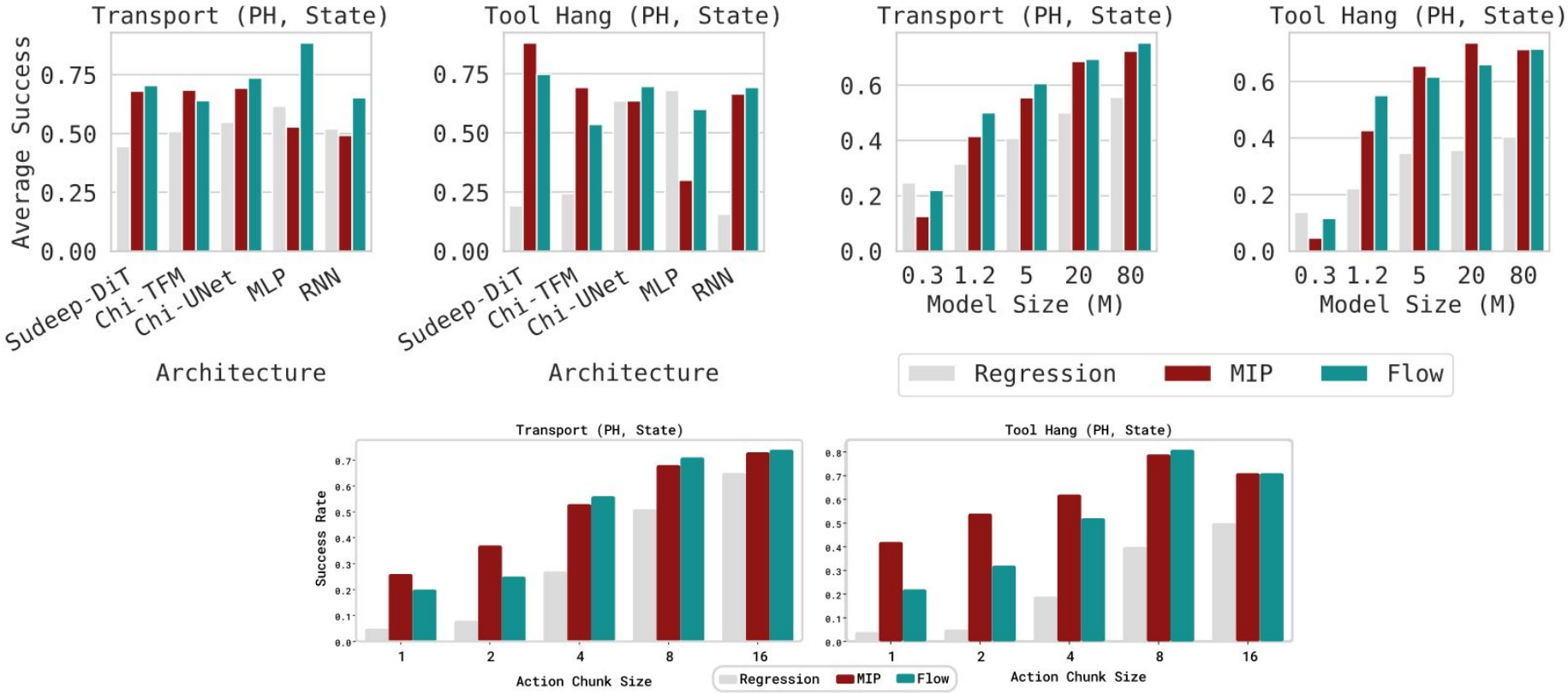


Compare: **Flow-Based** (GCP) Training to **Pure Regression**

improvements in performance concentrated only in **handful** of tasks.

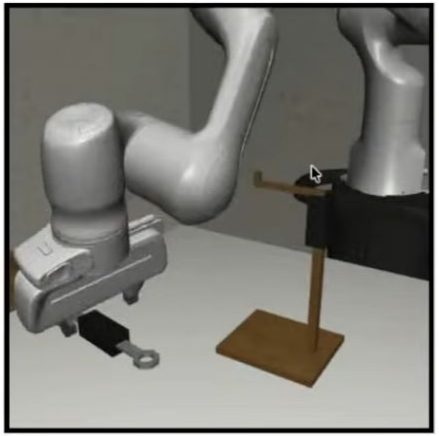
**Important: we controlled for architecture. Turns out regression with a "Diffusion Transformer" does great :)*

What else matter?



Are GCPs doing distribution learning?

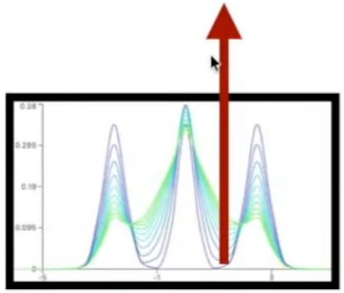
Q1: Are the improvements on multi-modal tasks?



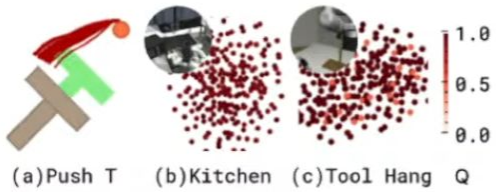
Finding 1: GCPs outperform regression only a **handful of tasks that require high-precision…**

Are GCPs doing distribution learning?

Finding 1: GCPs outperform RCPs only a **handful of tasks that require high-precision…**



Q2: Is **multi-modality even learned** on classically multi-modal tasks?



visualized action distributions exhibit **no discernible multi-modality**

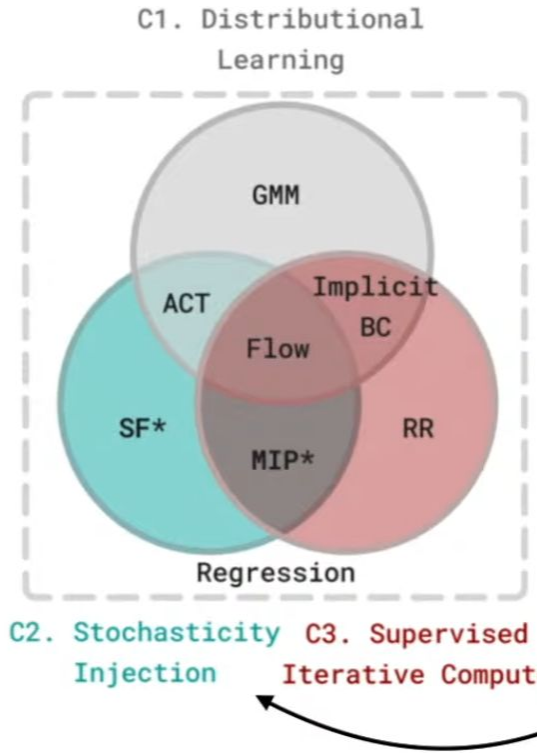
Task	$z \equiv 0$	$N(0, I)$	Mean z
Push-T	0.97	0.97	0.95
Kitchen	0.99	0.99	0.97
Tool-Hang	0.78	0.80	0.76

averaging over actions induces *negligible performance drop*

Dataset	Flow	Reg.
Original	0.78	0.58
Deterministic	0.72	0.64

on high-precision tasks, GCP > L2 regression even when expert labeled deterministically.

Can we match GCP without distribution learning?



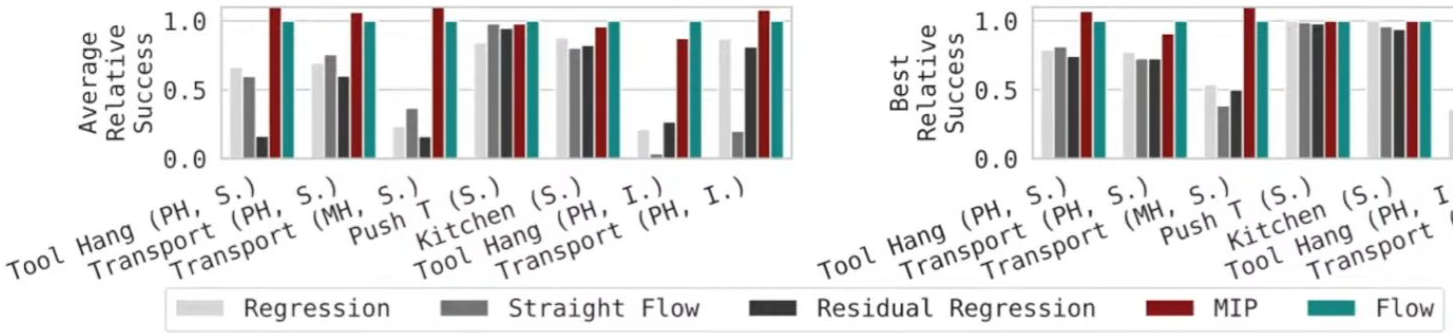
The Minimal Iterative Policy (MIP)

$$\hat{\pi}_0 \approx \arg\min_{\pi} \mathbb{E}^{\pi^\star} \|\pi(s) - a\|^2$$

$$\hat{\pi}_1 \approx \arg\min_{\pi} \mathbb{E}^{\pi^\star} \|\tilde{\pi}(s, a + \varepsilon z) - a\|^2$$

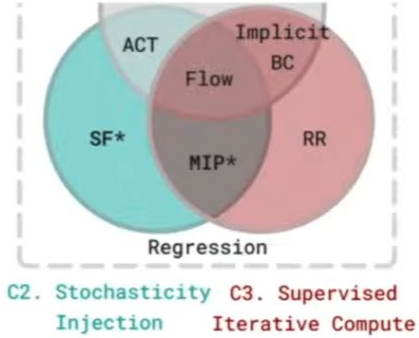
inference $\tilde{a} = \hat{\pi}_0(s), \quad a \leftarrow \pi_1(\tilde{a}; s)$

Can we match GCP without distribution learning?

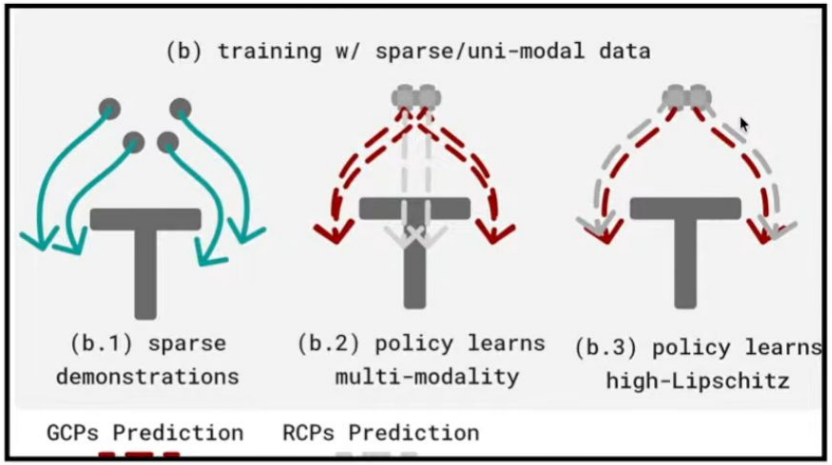


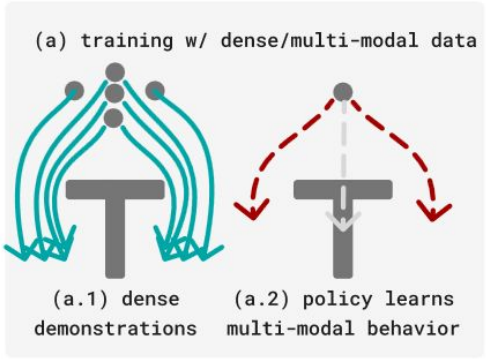
The Minimal Iterative Policy (MIP) performs well!

However, baselines with only C2/C3 **do not work...**

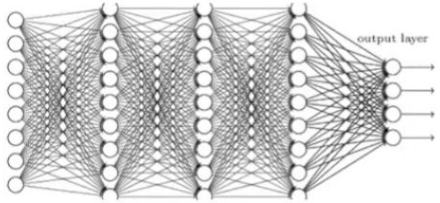


What are the benefits of iterative compute?



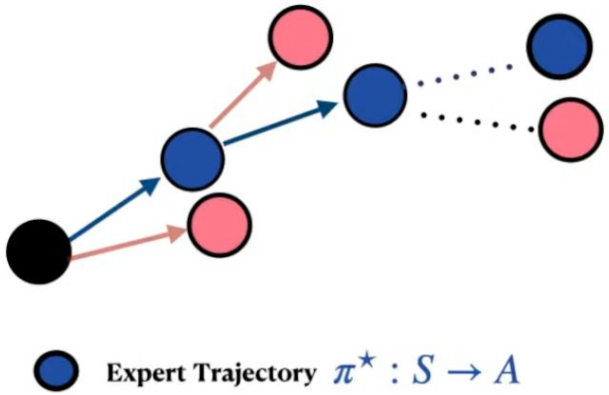


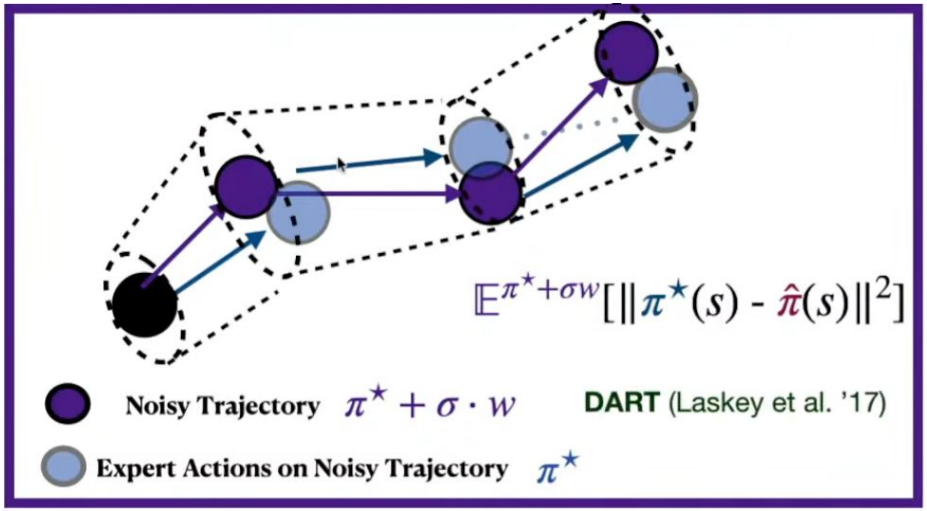
Hypothesis : the multi-step generation in flows represents a deeper network, thereby enabling higher-Lipschitz behavior?



False! Informal Theorem: In a canonical example of learning uni-modal conditional distributions (log-concave measures), learning a flow-model (even with **infinite discretization steps**) does not improve the ease of representing high-Lipschitz functions!

What are the benefits of noise injections?

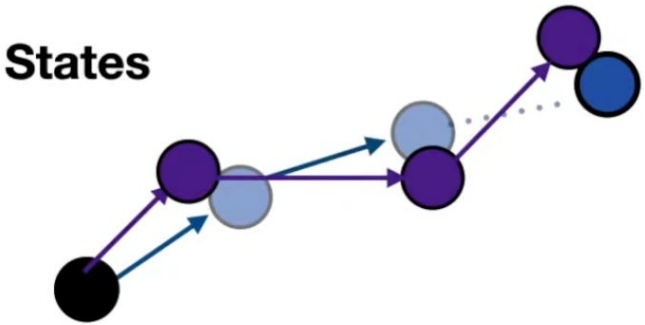


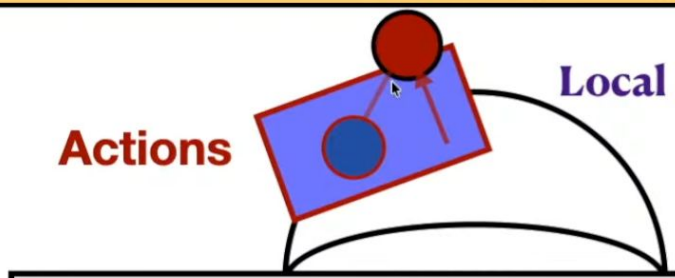


Theorem *(ZPPM**S**'25)**:** While BC cannot learn perfectly from expert data (**without open-loop stability**), being able to match the actions on **noisy trajectories** $\mathbb{E}^{\pi^\star+\sigma w}[\|\pi^\star(s) - \hat{\pi}(s)\|^2]$ is enough to have low imitation error.*

Exploratory Data Collection and Local Manifolds

Experiment: *Compare generalization on expert data v.s. noisy expert trajectories*





Dataset	Metric	Regression	SF	RR	MIP	Flow
Original	Off-manifold L_2	0.058	0.061	0.057	0.043	0.032
	Off-manifold L_1	0.072	0.073	0.071	0.057	0.046
	Validation L_2	0.073	0.071	0.062	0.069	0.074
	Validation L_1	0.110	0.106	0.124	0.104	0.116
Deterministic	Off-manifold L_2	0.067	0.063	0.062	0.054	0.042
	Off-manifold L_1	0.082	0.078	0.077	0.063	0.051
	Validation L_2	0.290	0.234	0.224	0.195	0.217
	Validation L_1	0.336	0.374	0.386	0.331	0.356

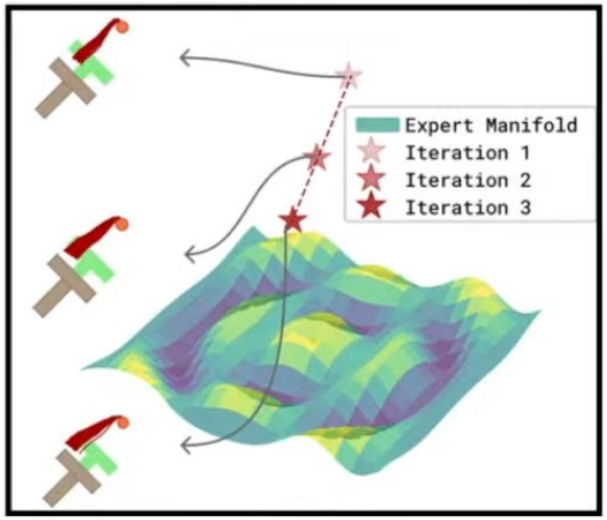
ground truth

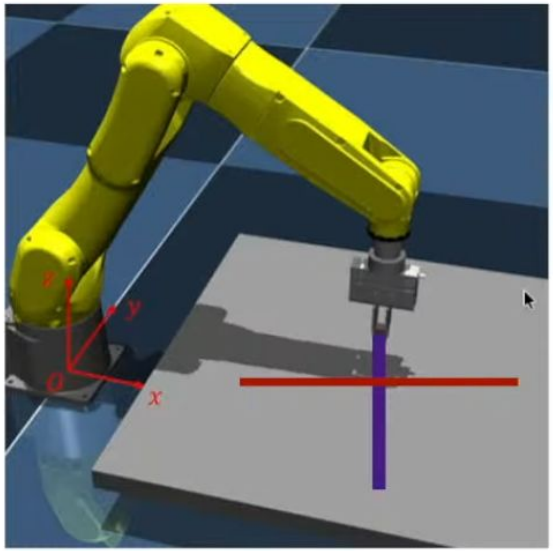
noisy counterfactual

action from the learner

Finding: GCPs have comparable validation loss on the distribution of expert data, but **exhibit lower off-manifold error on these noised-trajectories**

Not All Errors Created Equal





on-manifold error

off-manifold error

Iterative Computation + Stochasticity Project Onto the “Less-Bad” Error Directions, Simulating the Effect of Additional Data Being Collected!

Summary

In robotics, generative models have been proposed for their ability to approximate complex distributions.

However, other properties ('manifold adherence') appear to be the true mechanism driving performance.

I conjecture that manifold adherence emerges from the flow matching loss:

$$\hat{v} \approx \arg\min_{v} \mathbb{E}^{\pi^\star} \| v(ta + (1-t)z \mid t, \rho) - (z - a) \|^2,$$

Function approximation encourages smoothing in both generated + conditioning variable