Grounding Diffusion Models with Optimal Control

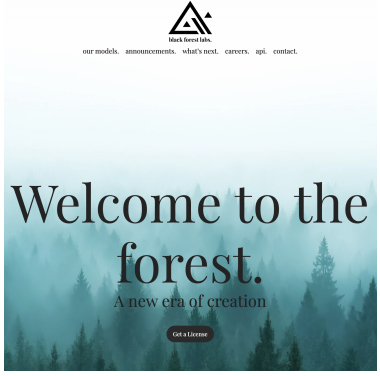
Jasper Hoffmann

Neurorobotics Group

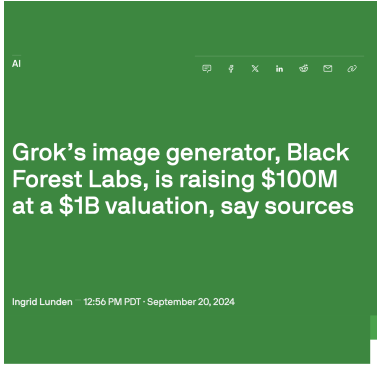
April 30, 2026

universität freiburg

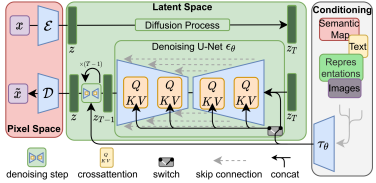
Freiburg Success Story: Black Forest Lab



[Source: www.blackforestlabs.ai]

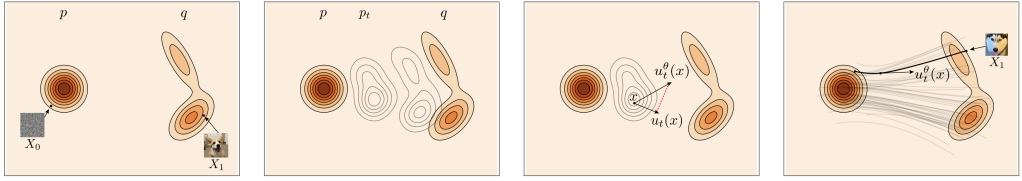


[Source: www.techcrunch.com]



[Source: Rombach et al. 2022]

Goals of this talk



(a) Data. **(b)** Path design. **(c)** Training. **(d)** Sampling.

[Source: Lipman et al. 2024]

Goals:

1. **Part I:** Mathematical introduction to flow matching
2. **Part II:** Establish diffusion as a key algorithmic component for control

Overview

Tutorial on Diffusion
- Normalizing Flows
- Flow Matching

Optimal Control with Diffusion Models
- Generative Planning
- Reinforcement Learning within World Models

Outlook

Overview

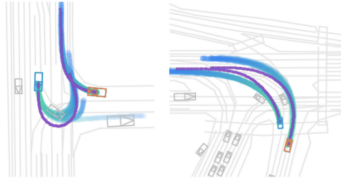
Generative Modelling

Given a dataset $\mathcal{D} \coloneqq \{Y_1, \dots, Y_n\} \subset \mathbb{R}^d$ from an unknown target distribution q.

Problem: How can we generate new samples distributed like q?

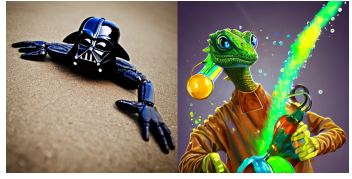
Remark: Conditional models generate samples from $q(\cdot \mid c)$ given context c. For simplicity, we omit the context during this talk.

Distribution over car trajectories



[Source: Jiang et al. 2023]

Distribution over images



[Source: github.com/CompVis/stable-diffusion]

Approach: "Noise to Sample" Transformation

Let p_0 be a simple base distribution on $\mathbb{R}^d$ (e.g. Gaussian), and $\varphi_\theta : \mathbb{R}^d \to \mathbb{R}^d$ a parameterized transformation.

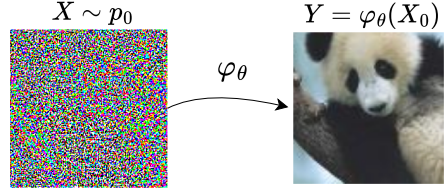
Goal: Learn θ so that the transformed samples follow the target distribution q.

Procedure:

- Sample $X \sim p_0$ and transform it via $Y = \varphi_\theta(X)$.
- This defines a new distribution p_θ:

$$p_\theta = (\varphi_\theta)_{\#} p_0.$$

- Optimize θ so that p_θ is close to q, thus $p_\theta \approx q$.



Maximum-Likelihood Formulation

Generative Modelling as Likelihood Maximization

Given the model $p_\theta = (\varphi_\theta)_\# p_0$, we optimize θ by maximizing the empirical log-likelihood:

$$\arg\max_{\theta \in \Theta} \mathbb{E}_{Y \sim \mathcal{D}} \big[\log p_\theta(Y)\big] \;=\; \arg\max_{\theta \in \Theta} \frac{1}{N} \sum_{i=1}^{N} \log p_\theta(Y_i)$$

- Increase likelihood of data $\mathcal{D}$ under model p_θ.
- We use $\mathcal{D} = \{Y_1, \ldots, Y_N\}$ with $Y_i \sim q$ to approximate:

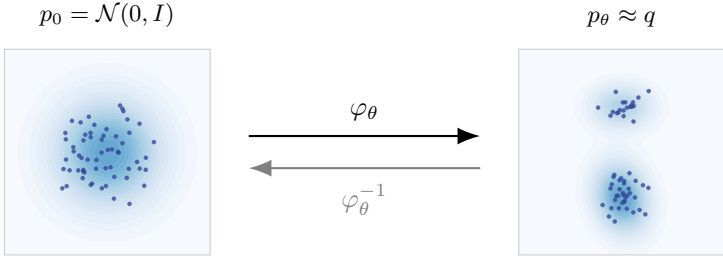
$$\mathbb{E}_{Y \sim q} \big[\log p_\theta(Y)\big] \;\approx\; \frac{1}{N} \sum_{i=1}^{N} \log p_\theta(Y_i)$$

- **Challenge:** To compute $\log p_\theta(y)$, we need a tractable expression for the density of the transformed distribution.

Normalizing Flow — Idea

Problem: The pushforward density $p_\theta = \varphi_{\theta\#}\, p_0$ has no closed-form expression for general φ_θ (non-invertible maps can fold space, making the density intractable).

Idea: Require φ_θ to be **invertible** and **continuously differentiable** (C^1-diffeomorphism).



(dots: samples, shading: density contours)

$\Rightarrow$ Invertibility allows transforming the density in *closed form*: map back via φ_θ^{-1}, evaluate p_0, correct for volume change — hence "*normalizing*."

Density Transformation

Theorem: Density Transformation

Let $\varphi : \mathbb{R}^d \to \mathbb{R}^d$ be invertible and continuously differentiable. If $X \sim p_X$, then $Y = \varphi(X)$ has density

$$p_Y(y) = \underbrace{p_X(\varphi^{-1}(y))}_{\text{preimage density}} \cdot \underbrace{\left|\det\left[\frac{\partial \varphi^{-1}}{\partial y}(y)\right]\right|}_{\text{volume correction}}$$

In our case: $X \sim p_0$ (base), $Y = \varphi_\theta(X) \sim p_\theta$ (model). Taking the log:

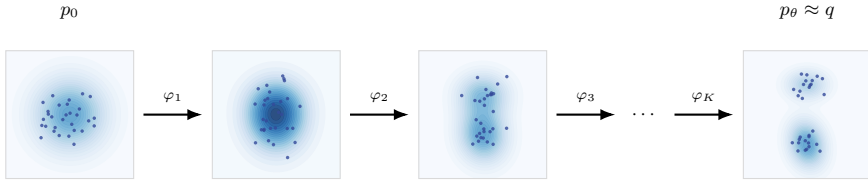
$$\log p_\theta(y) = \log p_0\big(\varphi_\theta^{-1}(y)\big) + \log\left|\det\left[\frac{\partial \varphi_\theta^{-1}}{\partial y}(y)\right]\right|$$

We can now directly optimize the empirical log-likelihood of the data:

$$\theta^\star = \arg\max_\theta \ \mathbb{E}_{Y\sim\mathcal{D}}\big[\log p_\theta(Y)\big] = \arg\max_\theta \ \sum_{i=1}^{N} \log p_\theta(Y_i)$$

Normalizing Flow — Residual Architecture

A popular approach: φ_θ is composed of K *residual blocks* $\varphi_k(x) = x + \delta\, u_k(x;\theta)$:



- Composition: $\varphi = \varphi_K \circ \ldots \circ \varphi_1$, each block φ_k is invertible if u_k is $1/\delta$-Lipschitz.
- Log-likelihood decomposes over blocks:

$$\log p_\theta(y) = \log p_0(\varphi^{-1}(y)) + \sum_{k=1}^{K} \log \left| \det \left[\frac{\partial \varphi_k^{-1}}{\partial x_{k+1}}(x_{k+1}) \right] \right|$$

From Residual Flows to Neural ODEs

- Rewrite the residual block $\varphi_k(x) = x + \delta\, u_k(x)$ as:

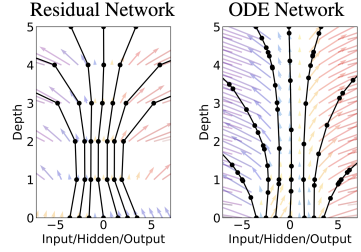
$$\frac{\varphi_k(x) - x}{\delta} = u_k(x)$$

- Set $\delta = 1/K$. As $K \to \infty$, the discrete composition $\varphi_K \circ \cdots \circ \varphi_1$ becomes a continuous process governed by an ODE:

$$\frac{\mathrm{d}x_t}{\mathrm{d}t} = \lim_{\delta \to 0} \frac{x_{t+\delta} - x_t}{\delta} = u_t(x_t)$$

- This motivates defining flows directly via a *velocity field* u_θ rather than composing discrete blocks.

Black circles represent evaluation locations.



[Source: Chen et al. 2018]

Residual Flow

- K residual layers $\varphi_k(x) = x + \delta\, u_k(x)$
- Inference: sequential forward pass

Continuous Normalizing Flow

- Learned velocity field $u_\theta(x, t)$
- Inference: integrate Neural ODE (Euler, adaptive schemes, ...)

Continuous Normalizing Flow — Definition

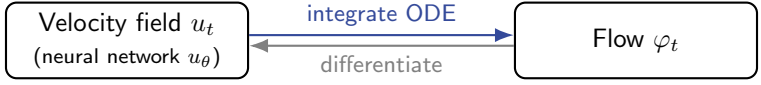
A *continuous normalizing flow* (CNF) is defined by a *velocity field* $u_t : \mathbb{R}^d \to \mathbb{R}^d$ via the ODE.

Definition: Initial Value Problem

$$\frac{\mathrm{d}}{\mathrm{d}t}\varphi_t(x) = u_t(\varphi_t(x)) \qquad \text{(ODE)}$$

$$\varphi_0(x) = x \qquad \text{(initial condition)}$$

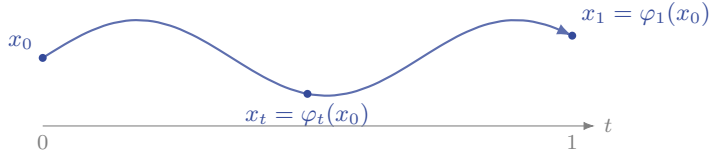
Existence: u_t Lipschitz $\Rightarrow$ unique solution of φ_t and is invertible.



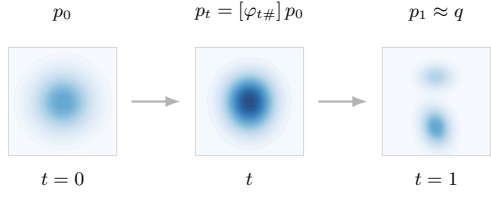
Key equivalence: Specifying velocity field $u_t \Leftrightarrow$ specifying flow φ_t.

Flows and Probability Paths

A flow φ_t maps each point x_0 to $x_t = \varphi_t(x_0)$:



Applied to a whole *distribution* $X_0 \sim p_0$, the flow induces a *probability path*:



We say u_t *generates* the probability path $(p_t)_{t\in[0,1]}$ if $X_t = \varphi_t(X_0) \sim p_t$ for all t.

Goal: Find u_θ such that $p_0 \xrightarrow{u_\theta} p_1 \approx q$.

Continuity Equation and Change of Variables

Continuity equation: p_t evolves according to

$$\frac{\mathrm{d}}{\mathrm{d}t} p_t(x) + \nabla \cdot (p_t\, u_t)(x) = 0 \quad \text{(conservation of mass)}$$

Instantaneous change of variables:

Along a trajectory $\varphi_t(x)$ one can show that:

$$\frac{\mathrm{d}}{\mathrm{d}t} \log p_t(\varphi_t(x)) = -\nabla \cdot u_t(\varphi_t(x))$$

Integrating from $t = 0$ to $t = 1$:

$$\log p_1(\varphi_1(x)) = \log p_0(x) - \int_0^1 (\nabla \cdot u_t)(\varphi_t(x))\, \mathrm{d}t$$

Substitution:

Let $y = \varphi_1(x)$, then $x = \varphi_1^{-1}(y)$ and

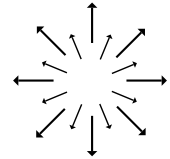
$$\log p_1(y) = \log p_0(\varphi_1^{-1}(y)) - \int_0^1 (\nabla \cdot u_t)(\varphi_t(\varphi_1^{-1}(y)))\, \mathrm{d}t$$

Divergence:

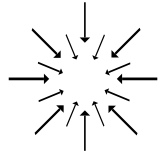
For $u : \mathbb{R}^d \to \mathbb{R}^d$

$$\nabla \cdot u(x) \coloneqq \sum_{i=1}^{d} \frac{\partial u_i(x)}{\partial x_i}$$

Intuition:



Spreading out: $\nabla \cdot u_\theta > 0$



Compressing: $\nabla \cdot u_\theta < 0$

Training Continuous Normalizing Flows

Objective: Maximum likelihood, same as before:

$$\theta^\star = \arg\max_{\theta \in \Theta} \mathbb{E}_{Y \sim \mathcal{D}} \big[\log p_1(Y; \theta) \big]$$

Challenge: Computing $\log p_1(Y; \theta)$ requires:

1. Solving the flow ODE *backwards* to find $x_0 = \varphi_1^{-1}(Y)$
2. Integrating the divergence $\nabla \cdot u_\theta$ along the trajectory

Approaches [Chen et al. 2018, Grathwohl et al. 2019]:

- Augmented ODE (jointly track state + log-density)
- Adjoint method for gradients (backprop through the ODE)
- Hutchinson trace estimator for $\nabla \cdot u_\theta$ in high dimensions

Problem: Training is *simulation-based* — requires solving the ODE. Expensive!

Summary so far

What we have:

- **Normalizing flows**: transform $p_0 \to q$ via bijective φ_θ; closed-form density via change of variables.
- **Continuous Normalizing Flows**: define φ_θ via a velocity field u_θ and the flow ODE; more flexible as u_θ needs to be only Lipschitz.

Remaining issues:

- Discrete NFs: architecture design for invertibility is restrictive.
- CNFs: training requires costly ODE simulation + divergence estimation.

Question: Can we train a CNF *without* integrating the ODE during training?
$\Rightarrow$ **Flow Matching** (simulation-free training)

Overview

Flow Matching

Flow matching is a simulation-free way to train continuous normalizing flows (CNFs). Remember the CNF φ_t is completely described by the velocity field u_θ!

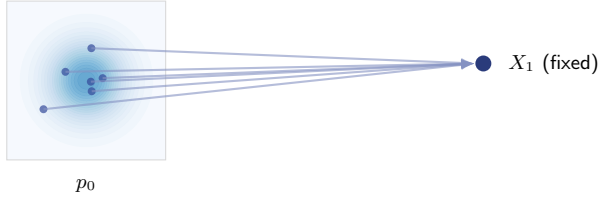
Idea: Given a target vector field u, train $\hat{u}_\theta$ to *match* u:

$$\mathcal{L}_{\mathrm{FM}}(\theta) = \mathbb{E}_{t\sim\mathcal{U}(0,1),\, X\sim p_t}\left[\left\|\hat{u}_\theta(t,X) - u(t,X)\right\|^2\right],$$

Problem: We need a target vector field to learn from...

Conditional Flow Matching — Idea

Observation: We don't know the marginal velocity field $u_t(x)$ that transforms $p_0 \to q$.
But for a *fixed* target sample X_1, the velocity field is trivial — just walk in a straight line!



Conditional probability path: Linear interpolation defines a path from p_0 to X_1:

$$X_t = (1-t)\,X_0 + t\,X_1, \quad X_0 \sim \mathcal{N}(0, I) \qquad \Rightarrow \qquad p_t(\cdot \mid X_1) = \mathcal{N}\big(t\,X_1,\ (1-t)^2 I\big)$$

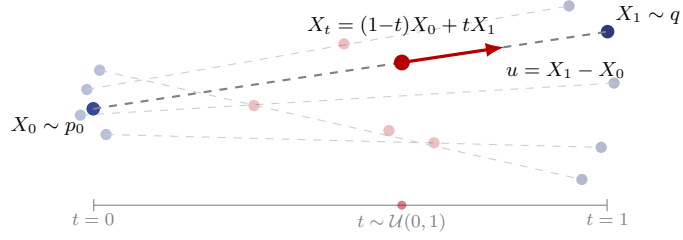
Conditional velocity field: The velocity along this path is simply:

$$u(t, X_t \mid X_1) = X_1 - X_0$$

Constant direction, straight to X_1, so no ODE required to construct it!

Conditional Flow Matching — Training

Key question: Where do we evaluate and learn the velocity field?



One training step:

1. Sample source $X_0 \sim p_0$, target $X_1 \sim q$ (from dataset), time $t \sim \mathcal{U}(0,1)$
2. Interpolate: $X_t = (1-t)\,X_0 + t\,X_1$
3. Learn: train $\hat{u}_\theta(t, X_t)$ to predict the direction $X_1 - X_0$

That's it! Each (X_0, X_1, t) triple gives one training sample, thus no simulation required.

Why does this work? — Marginalization

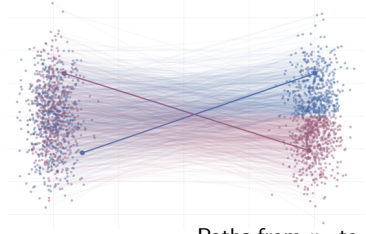
Our training loss (conditional flow matching):

$$\mathcal{L}_{\mathrm{CFM}}(\theta) = \underbrace{\mathbb{E}_{X_1 \sim q}}_{\substack{\text{average over} \\ \text{all targets}}} \Bigg[\underbrace{\mathbb{E}_{t,\, X_0 \sim p_0}}_{\substack{\text{for each target:} \\ \text{random time \& source}}} \Big[\big\| \hat{u}_\theta(t, X_t) - (X_1 - X_0) \big\|^2 \Big] \Bigg]$$

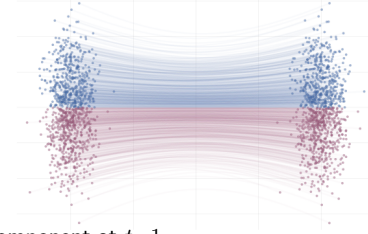
By averaging over all X_1, the network implicitly learns the *marginal* velocity field:

$$u_t(x) = \mathbb{E}_{X_1 \sim q}\big[u(t, x \mid X_1) \big]$$

Conditional flows (per target X_1)



Marginal flow (averaged over $X_1 \sim q$)



Paths from p_0 to q, coloured by sign of y-component at t=1.

Conditional Flow Matching — Objective

General form (works for any conditional path design):

$$\mathcal{L}_{\mathrm{CFM}}(\theta) = \mathbb{E}_{t\sim\mathcal{U}(0,1),\, X_1\sim q,\, X\sim p_t(\cdot|X_1)}\left[\left\|\hat{u}_\theta(t,X) - u(t,X \mid X_1)\right\|^2\right]$$

Our case (linear interpolation, Gaussian base):

$$\mathcal{L}_{\mathrm{CFM}}(\theta) = \mathbb{E}_{t\sim\mathcal{U}(0,1),\, X_1\sim q,\, X_0\sim\mathcal{N}(0,I)}\left[\left\|\hat{u}_\theta(t,X_t) - (X_1 - X_0)\right\|^2\right]$$

with $X_t = (1-t)\,X_0 + t\,X_1$.

Well-posedness: The conditional path design uniquely determines:

1. the probability path $p_t = \mathbb{E}_{X_1\sim q}[p_t(\cdot \mid X_1)]$,
2. the flow map φ_t (how each particle moves),
3. and hence a unique velocity field u_t via $u_t(\varphi_t(x)) = \frac{\mathrm{d}}{\mathrm{d}t}\varphi_t(x)$.

Flow Matching — A Simple Algorithm

Algorithm: Training

Require: Dataset $\mathcal{D} = \{Y_i\}_{i=1}^N$, network $\hat{u}_\theta$

repeat
  Sample $X_1 \sim \mathcal{D}$
  Sample $X_0 \sim \mathcal{N}(0, I)$
  Sample $t \sim \mathcal{U}(0, 1)$
  $X_t \leftarrow t\, X_1 + (1-t)\, X_0$
  $\theta \leftarrow \theta - \eta \nabla_\theta \left\| \hat{u}_\theta(t, X_t) - (X_1 - X_0) \right\|^2$
until converged

Key: No ODE integration needed!
Just regression on straight-line targets.

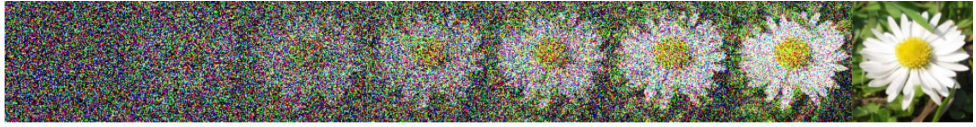
Algorithm: Inference

Require: Trained $\hat{u}_\theta$, steps K, $\delta = 1/K$

Sample $X_0 \sim \mathcal{N}(0, I)$
for $k = 0, \ldots, K-1$ **do**
  $t_k \leftarrow k/K$
  $X_{t_{k+1}} \leftarrow X_{t_k} + \delta\, \hat{u}_\theta(t_k, X_{t_k})$
end for
return $X_1 \approx$ new sample from q

Key: Euler integration of the learned velocity field from noise to data.

Generating Images

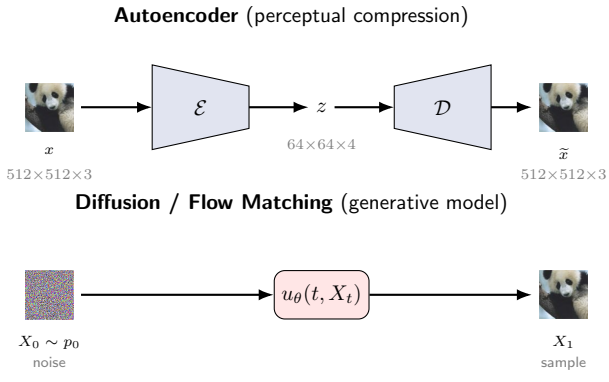


Integrating the learned velocity field $\hat{u}_\theta$ from noise (t=0) to data (t=1): each frame shows X_t at increasing t.

But how does this scale to 512×512 images? $\Rightarrow$ **Stable Diffusion**

Stable Diffusion — Idea

Problem: Running diffusion directly in pixel space ($512 \times 512 \times 3$) requires hundreds of GPU days.



Key idea [Rombach et al. 2022]: Separate *perceptual compression* from *generative modelling*.
$\Rightarrow$ First compress, then do diffusion on latent space z instead of pixel space x.

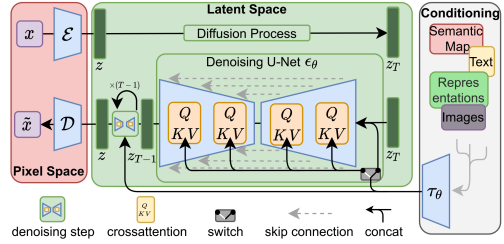
Result: Single-GPU training feasible instead of 150-1000 high-end GPUs.

Stable Diffusion — Architecture

Two-stage pipeline:

1. **Autoencoder** (trained once, frozen):
   $z = \mathcal{E}(x), \quad \hat{x} = \mathcal{D}(z)$
2. **Diffusion / Flow Matching** in latent space:
   Learn $u_\theta(t, z)$ on latents z
3. **Generation:**
   Sample z via denoising $\rightarrow$ decode $\hat{x} = \mathcal{D}(z)$

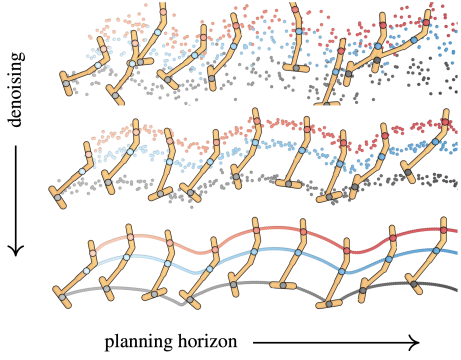
Conditioning: Text (CLIP/T5) via cross-attention.



[Source: Rombach et al. 2022]

Remark: Diffusion/flow matching (not necessarily latent) also used for protein structure (AlphaFold3), video generation, speech, robotics, ...

From Generating Images to Generating Plans



[Source: Janner et al. 2022]

Can we generate **plans** the same way we generate **images**?

Overview

From Generating Images to Generating Plans

We have seen how diffusion/flow matching generates *images*.

Key observation: The same framework can generate *many* high-dimensional structured object/distribution including **trajectories**, **plans**, and **stochastic dynamics**:

Image generation: $X_0 \sim p_0 \xrightarrow{u_\theta(\,\cdot\,|\text{text})}$ image

Plan generation: $X_0 \sim p_0 \xrightarrow{u_\theta(\,\cdot\,|x_0)} \tau = (x_0, a_0, \ldots, x_H)$

Learned dynamics: $X_0 \sim p_0 \xrightarrow{u_\theta(\,\cdot\,|x_k, a_k)} x_{k+1}$

Next: How is this leveraged for planning and control?

Diffusion for Control — Two Prominent Paradigms

Recall: A velocity field u_θ implicitly induces a distribution p_θ via the flow ODE. We write $p_\theta(\cdot \mid c)$ for the distribution conditioned on context c.

Two prominent paradigms:

1. **Generative Planner** [Janner et al. 2022, Ajay et al. 2023]. Sample trajectories based on historic dataset of trajectories:
$$\tau \sim p_\theta(\cdot \mid x_0) \quad \text{with } \tau = (x_0, a_0, \dots, x_H)$$

2. **Reinforcement Learning in World Models** [Hafner et al. 2025]. Learn dynamics $p_\theta(\cdot \mid x_k, u_k)$, but used as learned *simulator* to train a policy via synthetic rollouts.

Overview

Generative Planner — Idea

Object: Learn a conditional generative model over trajectories:

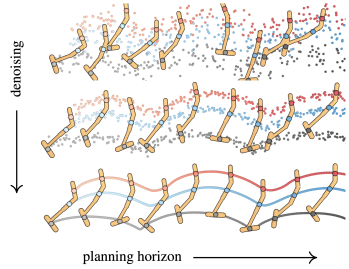
$$\tau \sim p_\theta(\cdot \mid x_0, c) \quad \text{with } \tau = (x_0, a_0, \ldots, x_H)$$

where c encodes context (goal, returns, constraints).

Planning as sampling:

1. Train p_θ on trajectory dataset $\mathcal{D}$
2. At test time, condition on current state x_0 and desired context c
3. Sample trajectory $\tau \sim p_\theta(\cdot \mid x_0, c)$
4. Execute first action a_0 from τ

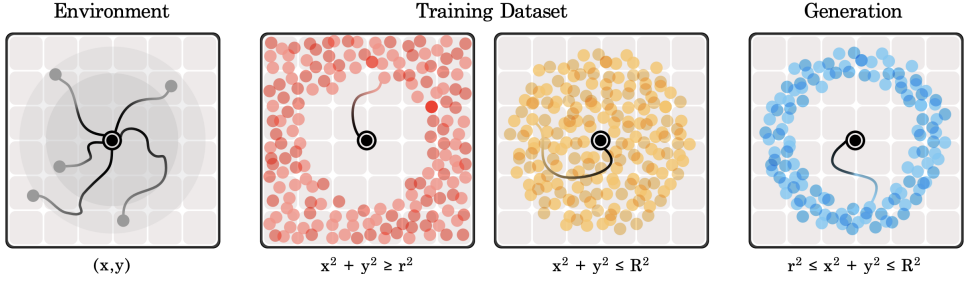
Key insight: No explicit dynamics model needed, as the generative model directly produces full trajectory plans.



[Source: Janner et al. 2022]

Generative Planner — Conditioning on Returns and Feasibility

Key idea: The dataset contains trajectories of varying quality. By conditioning, we can steer generation toward *high-return* and *feasible* trajectories.



[Source: Ajay et al. 2023]

How? Label each trajectory τ in $\mathcal{D}$ with properties c (return, constraint satisfaction, ...), then train a *conditional* model:

$$\tau \sim p_\theta(\cdot \mid x_0,\ c = \text{“high return, feasible”})$$

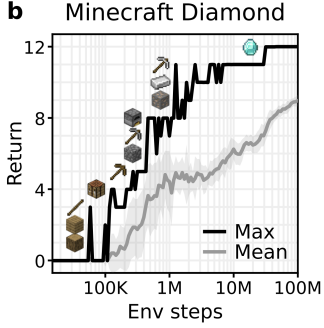
At test time, set c to the *desired* property $\Rightarrow$ generate only trajectories that satisfy it.

Overview

The Minecraft Diamond Challenge

Task: Starting with no tools, obtain a diamond from raw pixels and mouse/keyboard actions.

[First-person view — partial observability]



[Source: Hafner et al. 2024]

Challenges: partially observable, long horizon ($>$20k actions), sparse rewards, $\sim$20 sequential sub-tasks, procedural worlds, stochastic, offline only

World Model — Latent Diffusion Architecture (DreamerV4)

DreamerV4 does control from offline data using the following steps:

Step 1: Train Causal Tokenizer from Video Data (cf. autoencoder $\mathcal{E}/\mathcal{D}$ in Latent Diffusion)

- Let o_k be the k-th frame. Encoder $z_k = \mathcal{E}(o_{\leq k})$, decoder $\hat{o}_k = \mathcal{D}(z_k)$
- *Causal transformer:* Predict z_k such that it only depends on $o_{\leq k}$
- *Occlusion masking:* Remove parts of the image, to force reconstruction of $\hat{o}_k$ to rely on history $o_{<k}$ instead of just o_k.

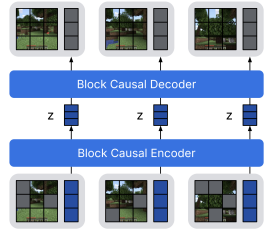
$\Rightarrow$ z_k summarizes the observation history: a reconstruction *belief state*

Step 2: Learn Latent Interactive Dynamics (cf. diffusion model u_θ, modelling implicitly p_θ.)

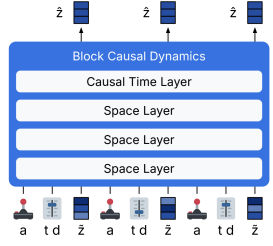
- Given a dataset of interaction data of form $\{(o_0, a_0, o_1, a_1, \dots)\}$
- Encode observations into latents $z_k = \mathcal{E}(o_{\leq k})$
- Learn conditional **diffusion model** p_θ in latent space:

$$z_{k+1} \sim p_\theta(\cdot \mid z_{\leq k}, a_{\leq k})$$

$\Rightarrow$ Diffusion enables stochastic and multimodal modelling



(a) Causal Tokenizer



(b) Interactive Dynamics

[Hafner et al. 2025]

World Model — Imagination Training

Step 1&2: Train encoder $\mathcal{E}$ and decoder $\mathcal{D}$ on recorded videos, latent dynamics p_θ on interactive datasets.

Step 3: Reinforcement Learning on latent Simulation:

1. **Policy initialization.** Learn a policy $\pi_\phi(\cdot \mid z_{\leq k}, a_{<k})$ and reward model $R_\phi(z_k)$ via behavioral cloning on the action-labeled data.
2. **Imagination training.** Fine-tune π_ϕ by RL *entirely inside the world model*:

$$\max_\phi \mathbb{E}_{\tau \sim p_\theta, \pi_\phi} \left[\sum_{k=0}^{H} \gamma^k R_\phi(z_k) \right]$$

   No environment interaction during policy optimization.

Result: DreamerV4 is the first agent to obtain diamonds in Minecraft from offline data alone.

For state-of-the-art RL read this paper (shortcut forcing objective, architecture, RL loss) in Hafner et al. 2025.

Overview

Overview

Recent work on combining first-principle MPC and Diffusion

Recent work:

- **Warmstart MPC**: Imitate MPC with a diffusion model, then warm-start, can also help against local minima [Huang et al. 2025]
- **Feasible Dynamics**: Regularize diffusion with a dynamics model. Add penalization flow if dynamics equality constraints are not satisfied [Unpublished work]
- **Feasible Planning**: During generation, iteratively project planned trajectory into feasible region using an MPC safety layer [Xiao et al. 2023]

Key question: How to get the best of both worlds?

- Diffusion: expressive, multi-modal trajectory distributions
- MPC: constraint satisfaction, stability guarantees, optimality, explicit cost optimization

Application: Predictive Control under Exogenous Uncertainty

Setting: Building energy management (heat pump control).

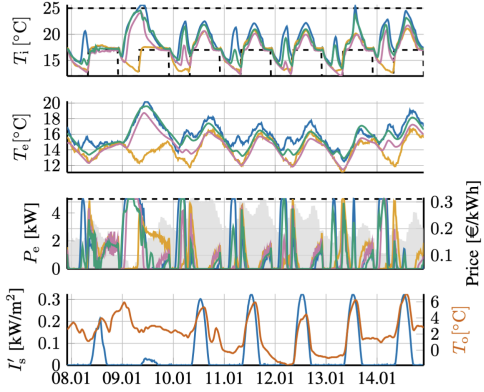
- **State** x: indoor / envelope temperatures
- **Control** u: heat pump electrical power
- **Exogenous** w: weather, solar irradiance, electricity prices, occupant heat gains

Key challenge:
The exogenous inputs $w_{0:N-1}$ are *uncertain* and *multimodal* (e.g., cloud cover).

$\Rightarrow$ Learn a *generative forecast model* via diffusion:

$$w_{0:N-1}^{(i)} \sim p_\theta(\,\cdot \mid s, c\,)$$

where s = current state, c = context (time of year, recent history).



Closed-loop heat pump control:
temperatures, power, prices, weather.

A first simple idea? — Scenario-based Stochastic MPC

Model with sampled disturbance scenarios:

$$x_{k+1} = f(x_k, u_k, w_k), \qquad w_{0:N-1} \sim p_\theta(\cdot \mid s, c)$$

Scenario OCP: Given M forecast scenarios from the diffusion model, solve:

$$\begin{aligned}
\min_{u_0,\, x_1^{(i)}, u_1^{(i)}, \ldots, u_{N-1}^{(i)}, x_N^{(i)}} \quad & \frac{1}{M} \sum_{i=1}^{M} \left[V(x_N^{(i)}) + \sum_{k=0}^{N-1} l(x_k^{(i)}, u_k^{(i)}) \right] \\
\text{s.t.} \quad & x_0^{(i)} = s, && 1 \le i \le M, \\
& u_0^{(i)} = u_0, && 1 \le i \le M, \\
& x_{k+1}^{(i)} = f(x_k^{(i)}, u_k^{(i)}, w_k^{(i)}), && 1 \le i \le M,\ 0 \le k \le N-1, \\
& 0 \le h(x_k^{(i)}, u_k^{(i)}), && 1 \le i \le M,\ 0 \le k \le N-1
\end{aligned}$$

Only u_0 is shared, future controls $u_{k\ge1}^{(i)}$ adapt per scenario.

Details to be worked out!

Diffusion: A New Algorithmic Component for Control

What diffusion gives us:

- A *generative model*: given a dataset, produce new high-fidelity samples
- Captures complex, multimodal distributions (not just mean predictions)
- Flexible conditioning: $p_\theta(\cdot \mid \text{state}, \text{context}, \text{goal}, \ldots)$
- Scalable to high-dimensional, structured outputs (trajectories, images, sequences)

For the control community, this means:

- Learned dynamics that are *stochastic and multimodal* — not just a single next-state prediction
- Scenario generation for stochastic MPC from data, without hand-crafted noise models
- Generative planners that produce diverse, constraint-aware trajectory distributions
- World models that imagine futures for policy optimization
- Maybe more...

Diffusion models are not a replacement for MPC —
they are a new *building block* that complements first-principles control.

Further Material

Part I — Flow Matching:

- Flow matching guide and coding framework by Meta: `https://arxiv.org/abs/2412.06264`
- Blog post by Machine Learning Group of Cambridge: `https://mlg.eng.cam.ac.uk/blog/2024/01/20/flow-matching.html`
- Flow matching introduction by Yannick Kilcher: `https://www.youtube.com/watch?v=7NNxK3CqaDk`

Part II — Diffusion for Control:

- Diffuser: Janner et al. 2022 — *Planning with Diffusion for Flexible Behavior Synthesis*
- Decision Diffuser: Ajay et al. 2023 — *Is Conditional Generative Modeling all you need for Decision-Making?*
- Dreamer4: Hafner et al. 2025 — *Training Agents Inside of Scalable World Models*

Thank you for your attention!

Generative Planner — Conditioning Mechanisms

1. **Classifier-free guidance** (train-time conditioning):
   Train with and without labels c; at inference, amplify the conditional signal:
   $$\tilde{u}_\theta(t, \tau) = u_\theta(t, \tau) + w\big[u_\theta(t, \tau \mid c) - u_\theta(t, \tau)\big]$$

2. **Inpainting / replacement** (test-time):
   Fix known values (e.g. current state x_0, goal x_H) — overwrite entries at each denoising step.

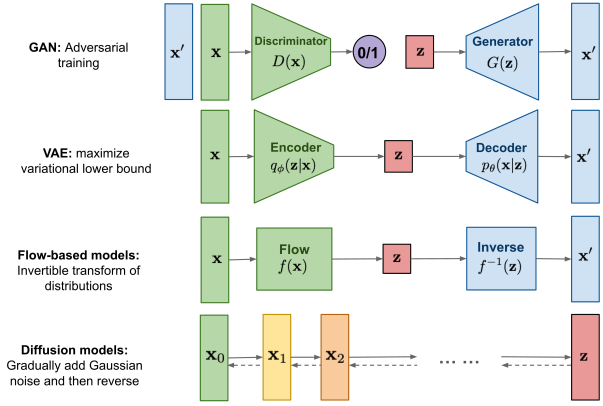
3. **Classifier guidance** (test-time, no retraining):
   Given any differentiable objective $J(\tau)$, nudge each iterate toward higher reward:
   $$\tilde{\tau}^{(i)} = \tau^{(i)} + \alpha \, \nabla_\tau J(\tau)\big|_{\tau=\tau^{(i)}}$$

Deep Generative Models

Generative models: Many different approaches to learning a generative model exist.

Focus of this talk:

- Normalizing flow models
- Continuous-time normalizing flows
- Flow matching models (diffusion)

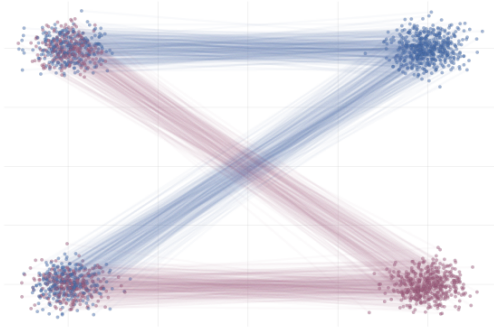


[Credit: nvidia]

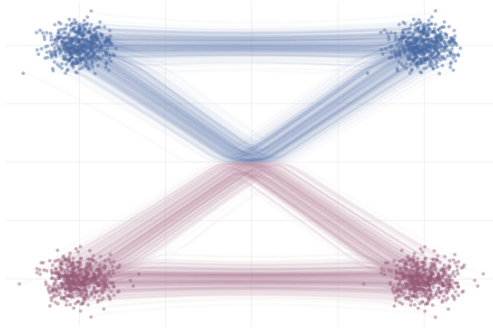
Latent variable z sampled from simple source distribution p_0, thus $z \sim p_0$

Example: Optimality?

Conditional velocity field $u_t(x \mid x_1)$



Marginal velocity field $u_t(x)$



Question: Is the marginal velocity field suboptimal? No!

Proof: Invertibility of Residual Blocks

Claim: $\varphi_k(x) = x + \delta\, u_k(x)$ is invertible if u_k is L-Lipschitz with $\delta L < 1$.

Proof (injectivity): Suppose $\varphi_k(x) = \varphi_k(x')$. Then:

$$\begin{aligned} x + \delta\, u_k(x) &= x' + \delta\, u_k(x') \\ \|x - x'\| &= \delta\, \|u_k(x') - u_k(x)\| \leq \delta L\, \|x - x'\| \end{aligned}$$

Since $\delta L < 1$, this implies $\|x - x'\| = 0$, thus $x = x'$. ✓

Proof (surjectivity): For fixed y, solve $\varphi_k(x) = y$, i.e. find $x = y - \delta\, u_k(x)$.
Define $T(x) = y - \delta\, u_k(x)$. Then:

$$\|T(x) - T(x')\| = \delta\, \|u_k(x) - u_k(x')\| \leq \delta L\, \|x - x'\|$$

Since $\delta L < 1$, T is a contraction $\Rightarrow$ Banach fixed point theorem gives unique fixed point. □