Flow Matching for Generative Modeling

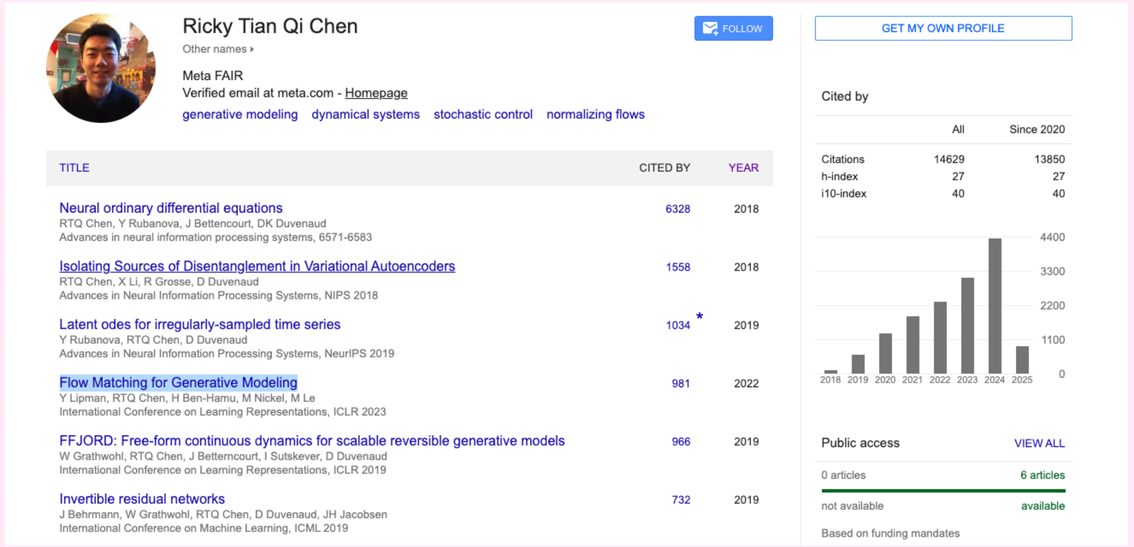
CSCI 699: Probabilistic and Generative Models

Oliver Liu, March 14

Some Background…

Why am I doing this…

- Got an interview with this guy…
- Knew little about flow matching 😨
- So I studied for a week 💦
- Another guy interviewed me 😅
- Told Willie I’d like to do something about flow matching
- Flow Matching Guide and Code



Ricky Tian Qi Chen

FOLLOW

GET MY OWN PROFILE

Other names ▸

Meta FAIR

Verified email at meta.com - Homepage

generative modeling dynamical systems stochastic control normalizing flows

TITLE	CITED BY	YEAR
Neural ordinary differential equations RTQ Chen, Y Rubanova, J Bettencourt, DK Duvenaud Advances in neural information processing systems, 6571-6583	6328	2018
Isolating Sources of Disentanglement in Variational Autoencoders RTQ Chen, X Li, R Grosse, D Duvenaud Advances in Neural Information Processing Systems, NIPS 2018	1558	2018
Latent odes for irregularly-sampled time series Y Rubanova, RTQ Chen, D Duvenaud Advances in Neural Information Processing Systems, NeurIPS 2019	1034 *	2019
Flow Matching for Generative Modeling Y Lipman, RTQ Chen, H Ben-Hamu, M Nickel, M Le International Conference on Learning Representations, ICLR 2023	981	2022
FFJORD: Free-form continuous dynamics for scalable reversible generative models W Grathwohl, RTQ Chen, J Betterncourt, I Sutskever, D Duvenaud International Conference on Learning Representations, ICLR 2019	966	2019
Invertible residual networks J Behrmann, W Grathwohl, RTQ Chen, D Duvenaud, JH Jacobsen International Conference on Machine Learning, ICML 2019	732	2019

Cited by

	All	Since 2020
Citations	14629	13850
h-index	27	27
i10-index	40	40

Public access VIEW ALL

0 articles 6 articles

not available available

Based on funding mandates

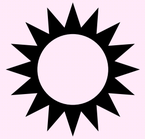
Normalizing Flows

A quick tour d'horizon

- High-level idea: **learn a map** from a simple distribution to a complex one.

Source (simple) distribution $Z \sim p \rightarrow X \equiv g_\theta(Z) \sim q$ Target (data) distribution

- For example, a sensor records brightness value Z, but the physical process of capturing the light is a nonlinear transformation $X = g(Z)$

Latent   Observed

Z $X = g(Z)$

$$q(X) = p(Z)\left|\frac{dZ}{dX}\right| = p(Z)\left|\frac{dg^{-1}(X)}{dX}\right|$$

Normalizing Flows

A quick tour d'horizon

- High-level idea: **learn a map** from a simple distribution to a complex one.

Source (simple) distribution $Z \sim p \rightarrow X \equiv g_\theta(Z) \sim q$ Target (data) distribution

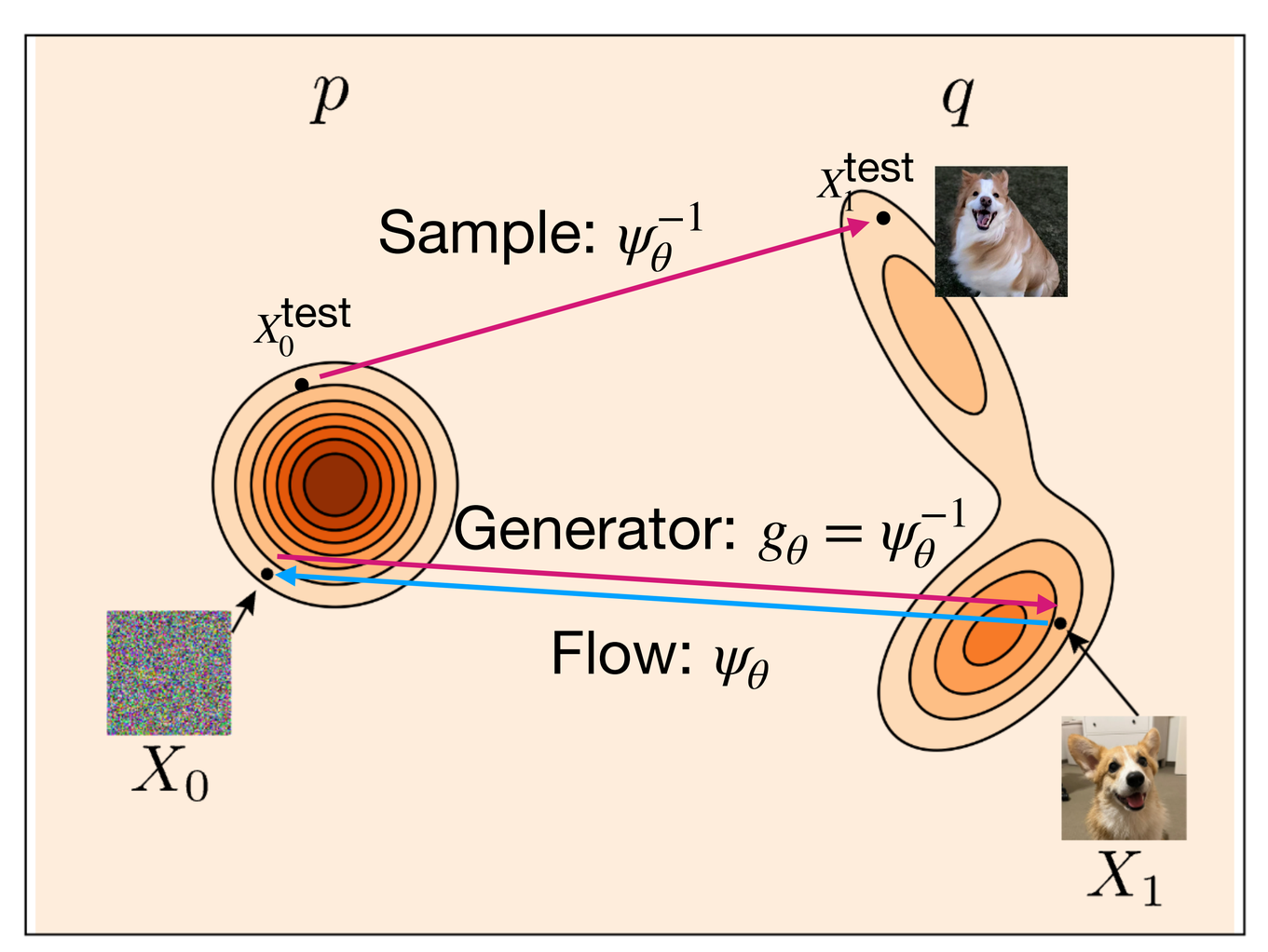
- If is g_θ endowed with an inverse ψ_θ ("flow"), then we can apply the change of variable formula to obtain:

$$
\begin{aligned}
q(x) &= p(\psi_\theta(x)) \, |\det \mathbf{J}\psi_\theta(x)| \\
&= p(\psi_\theta(x)) \, |\det \mathbf{J}g_\theta(\psi_\theta(x))|^{-1}
\end{aligned}
$$

- If g_θ (the "generator") is a powerful enough approximator then we can hopefully learn any complex distribution q

Normalizing Flows

A quick tour d'horizon



Continuous Normalizing Flows

A quick tour d'horizon

- **The Optimization Problem:**

$$\log q(x) = \log p(\psi_\theta(x)) + \log \left| \det \mathbf{J} g_\theta(\psi_\theta(x)) \right|^{-1}$$

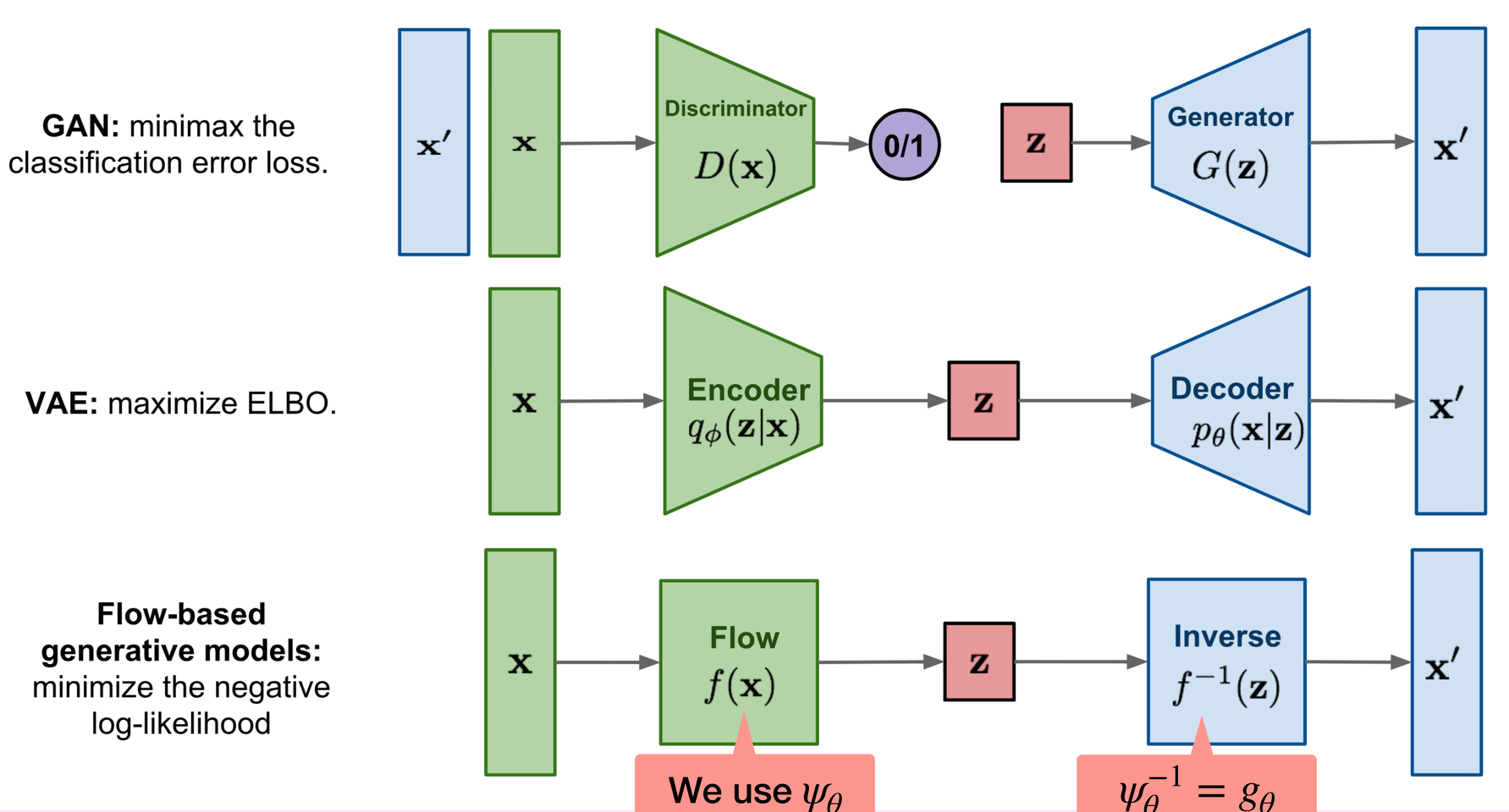
Sample data, and feed it into the flow model

Feed the transported data into the generator for reconstruction

- **Note:** ψ_θ And $g_\theta \equiv \psi_\theta^{-1}$ are parameterized with the same parameters θ

GAN vs. VAE vs. CNF

https://lilianweng.github.io/posts/2018-10-13-flow-models/



Why is Normalizing Flow Challenging?

$$\log q(x) = \log p(\psi_\theta(x)) + \log |\det \mathbf{J} g_\theta(\psi_\theta(x))|^{-1}$$

- The transport map (i.e. neural nets) $\psi_\theta : \mathbb{R}^d \to \mathbb{R}^d$ needs to be bijective.
- Needs to compute and (possibly) store the $\mathbb{R}^{d \times d}$ Jacobian matrix.
- For ImageNet: $d = 3 \times 224^2$!
- Normalizing flow has limited model parameterization and lacks efficiency.

Introducing Flow Matching

- **Same idea:** Transport a simple distribution p_0 to a target distribution $q = p_1$ via a time-dependent probability path $(p_t)_{0 \leq t \leq 1}$.
- **Core technique:** Use the probability path to alleviate both the bijection and Jacobian restriction.
- **Additional benefit:** Admits a simple optimization objective (that encapsulates diffusion) while providing a smoother sample path.

The Math

Let's verify step-by-step

- **Goal:** We'd like to define $(p_t)_{0 \le t \le 1}$ that has desirable properties.
- **One idea:**

$$p_t(x) = \int p_{t|1}(x \mid x_1) q(x_1) dx_1$$

Unknown real world data distribution

Conditional probability path

- Notice that this is [illegible]y, as each integrand is nonnegative, and by Fubini's Theorem

$$\begin{aligned}
\int p_t(x)dx &= \int \left(\int p_{t|1}(x \mid x_1) q(x_1) dx_1 \right) dx \\
&= \int \int p_{t|1}(x \mid x_1) q(x_1) dx dx_1 \\
&= \int q(x_1) \left(\int p_{t|1}(x \mid x_1) dx \right) dx_1 \\
&= \int q(x_1) \cdot 1 \cdot dx_1 \\
&= \int q(x_1) dx_1 \\
&= 1
\end{aligned}$$

The Math

Conditional Probability Path

This is only one way (or the simplest way) to design the probability path.

- $p_{t|1}(x \mid x_1)$ can be any arbitrary distribution. Which one is the best?
- A “boundary condition”: the final conditional should center around the datum.

$$p_{t|1}(x \mid x_1) \sim \mathcal{N}(x \mid tx_1, (1 - t^2)I)$$

a.k.a linear path

- This way, the marginal at $t = 1$ approximates the data distribution

$$p_1(x) = \int p_{1|1}(x \mid x_1) q(x_1) dx_1 \approx q(x)$$

Essentially the Dirac measure at x

The Math

The Linear Conditional Probability Path

- Sampling from $p_t(x) = \int p_{t|1}(x \mid x_1) q(x_1) dx_1$ with the linear path is very nice:

$$X_t = tX_1 + (1-t)X_0 \sim p_t$$

We won't show this, but it's not to hard to convince yourself that this is the case.

- **A subtle point:** sampling X_t requires sampling X_1 from the data distribution, but p_t does not depend on X_1, as it is marginalized!

The Math

Coming back to flows

- Early on, we termed the map from data to noise as $(\psi_t)_{t\in[0,1]} \equiv (p_t)_{t\in[0,1]}$.
- In the previous slides, we can write their integral forms.
- We'd like to parameterize a neural net to approximate this distribution.
- Can we directly solve the following objective?

$$\mathrm{Div}(\psi_t^\theta(X_t), \psi_t(X_t))$$

- Not really, since we do not know the data distribution $q(x)$, nor can we easily normalize the approximate density $\psi_t^\theta(X_t)$.

The Math

A compromise

- We will instead model the velocity field $(u_t)_{t\in[0,1]}$ of the flow $(\psi_t)_{t\in[0,1]}$.
- The velocity field $u : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$ determines the direction and speed that a sample point x is moving through the density at time t.
- The flow ψ and the velocity field u are tied by the continuity equation:

$$\frac{\partial \psi_t}{\partial t} + \nabla \cdot (\psi_t u_t) = 0$$

- Approximate u_t well $\Longrightarrow$ Approximate ψ_t well.

The Math

The objective function

$$\mathscr{L}_{\text{FM}}(\theta) = \mathbb{E}_{t,X_t} || u_t^\theta(X_t) - u_t(X_t) ||^2, \text{ where } t \sim \mathscr{U}[0,1], X_t \sim p_t$$

- Sadly, u_t is intractable for the exact same reason…
- But a conditional version of $\mathscr{L}_{\text{FM}}$ is **tractable**!
- Recall that $X_t = tX_1 + (1-t)X_0 \sim p_t$
- Then $X_{t|1} = tx_1 + (1-t)X_0 \sim p_{t|1}(\,\cdot\,|X_1 = x_1) = \mathcal{N}(\,\cdot\,|tx_1, (1-t)^2 I)$
- (With some math) the conditional velocity field is $u_t(x\,|\,x_1) = (x_1 - x)/(1-t)$

The Math

The conditional objective

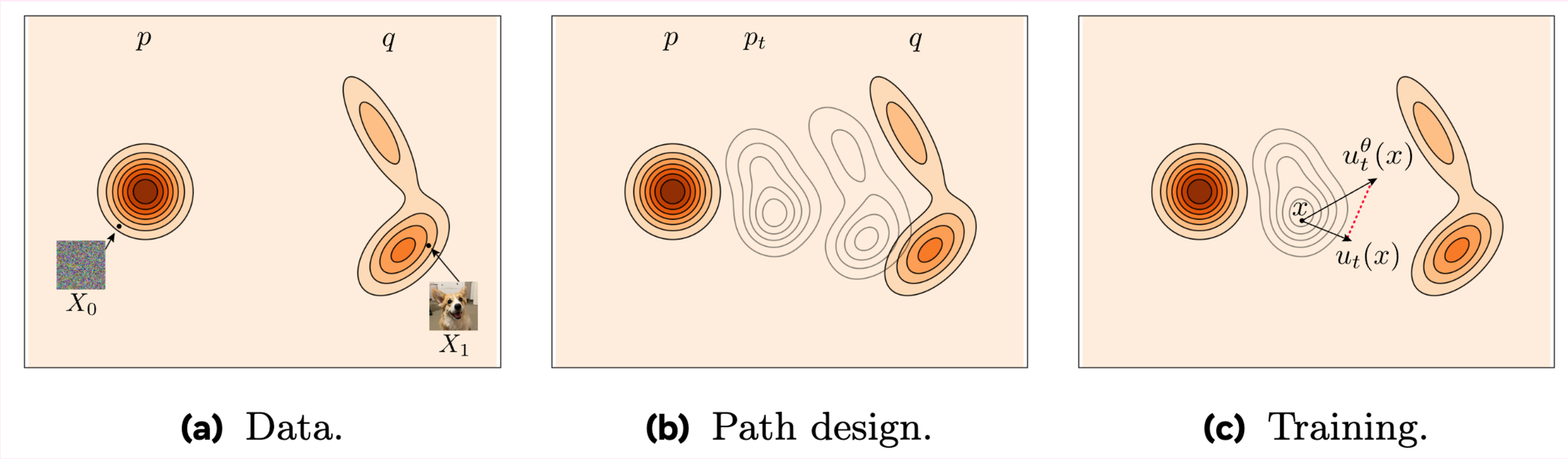
$$\mathscr{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t,X_t,X_1} \left|\left| u_t^{\theta}(X_t) - u_t(X_t \mid X_1) \right|\right|^2, \text{ where } t \sim \mathscr{U}[0,1], X_t \sim p_{t|1}, X_1 \sim q$$

- But $\mathscr{L}_{\text{FM}}$ is what we'd really hope to optimize, not its sample-conditioned variant. So what to do now?
- **Theorem 2 (Lipman et al. 2023):** *Assuming that $p_t(x) > 0$ for all $x \in \mathbb{R}^d$ and that $t \in [0,1]$, then, up to a constant independent of θ, $\mathscr{L}_{CFM}$ and $\mathscr{L}_{FM}$ are equal. Hence $\nabla_\theta \mathscr{L}_{FM}(\theta) = \nabla_\theta \mathscr{L}_{CFM}(\theta)$.*

  Proof Sketch: Apply the bilinear property of 2-norm, then apply Fubini's.

The Math

A Recap



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

The Math

Can we finally train a network now?

$$\mathscr{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t,X_t,X_1} \left|\left| u_t^\theta(X_t) - u_t(X_t \,|\, X_1)) \right|\right|^2, \text{ where } t \sim \mathscr{U}[0,1], X_t \sim p_{t|1}, X_1 \sim q$$

- Sample a time step $t \sim \mathscr{U}[0,1]$
- Draw a training sample $X_1 \sim q$
- Sample x from $X_t \sim \mathscr{N}(\,\cdot \,|\, tx_1, (1-t)^2 I)$
- Feed x into a neural net to compute $u_t^\theta(x)$
- Compute regression target $u_t(x \,|\, x_1) = (x_1 - x)/(1-t)$.
- Regress, and minimize the MSE!

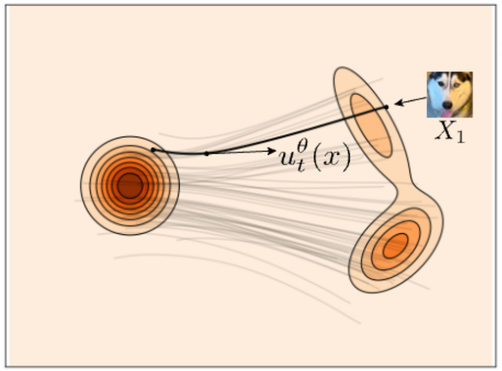
The Code

It’s embarrassingly simple!

This network should really be called VelocityField

```
class Flow(nn.Module):
  def __init__(self, dim: int = 2, h: int = 64):
    super().__init__()
    self.net = nn.Sequential(
      nn.Linear(dim + 1, h), nn.ELU(),
      nn.Linear(h, h), nn.ELU(),
      nn.Linear(h, h), nn.ELU(),
      nn.Linear(h, dim))

  def forward(self, x_t: Tensor, t: Tensor) -> Tensor:
    return self.net(torch.cat((t, x_t), -1))
```

```
:: Tensor, t_end: Tensor) -> Tensor:
and(x_t.shape[0], 1)
: ODE solver in this example
self(x_t + self(x_t, t_start) * (t_end - t_start) / 2,
```



(d) Sampling.

Let the wind (velocity field) takes you to wherever it takes you

Training

```
# training
flow = Flow()
optimizer = torch.optim.Adam(flow.parameters(), 1e-2)
loss_fn = nn.MSELoss()

for _ in range(10000):
  x_1  = Tensor(make_moons(256, noise=0.05)[0])
  x_0  = torch.randn_like(x_1)
  t    = torch.rand(len(x_1), 1)
  x_t  = (1 - t) * x_0 + t * x_1
  dx_t = x_1 - x_0
  optimizer.zero_grad()
  loss_fn(flow(x_t, t), dx_t).backward()
  optimizer.step()
```

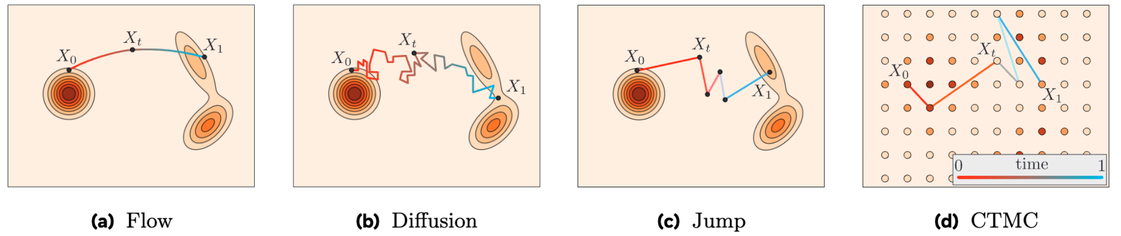
Sampling

```
# sampling
x = torch.randn(300, 2)
n_steps = 8
fig, axes = plt.subplots(1, n_steps + 1, figsize=(30, 4), sharex=True, sharey=True)
time_steps = torch.linspace(0, 1.0, n_steps + 1)

axes[0].scatter(x.detach()[:, 0], x.detach()[:, 1], s=10)
axes[0].set_title(f't = {time_steps[0]:.2f}')
axes[0].set_xlim(-3.0, 3.0)
axes[0].set_ylim(-3.0, 3.0)

for i in range(n_steps):
  x = flow.step(x, time_steps[i], time_steps[i + 1])
  axes[i + 1].scatter(x.detach()[:, 0], x.detach()[:, 1], s=10)
  axes[i + 1].set_title(f't = {time_steps[i + 1]:.2f}')
```

The Applications

It's a general modeling framework

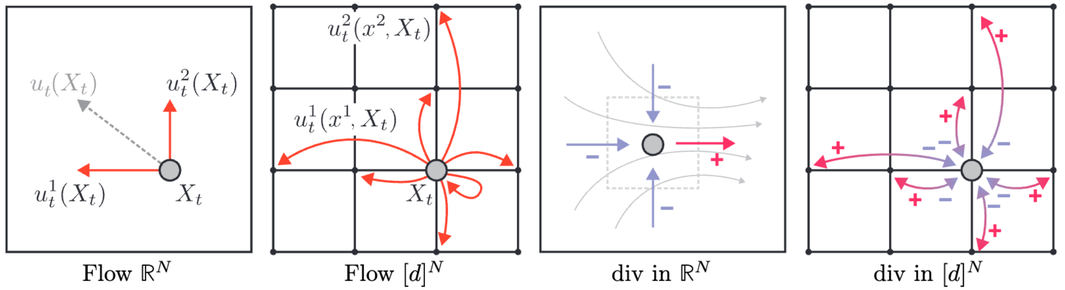
- Define p and q;
- Define flow paths $(\psi_t)_{t\in[0,1]} \equiv (p_t)_{t\in[0,1]}$;
- Derive VF $(u_t)_{t\in[0,1]}$ for $(\psi_t)_{t\in[0,1]}$;
- Implement a VF net:
  $u_t^\theta(X_t) : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$
- Optimize $\mathscr{L}_{\text{CFM}}(\theta)$



(a) Flow **(b)** Diffusion **(c)** Jump **(d)** CTMC

Figure 1 Four time-continuous processes $(X_t)_{0\le t\le 1}$ taking source sample X_0 to a target sample X_1. These are a flow in a continuous state space, a diffusion in continuous state space, a jump process in continuous state space (densities visualized with contours), and a jump process in discrete state space (states as disks, probabilities visualized with colors).

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



Flow $\mathbb{R}^N$ Flow $[d]^N$ div in $\mathbb{R}^N$ div in $[d]^N$

Figure 2 Discrete flow in $\mathcal{D} = [d]^N$ with $d = 4, N = 2$ (middle-left) versus continuous flow in $\mathbb{R}^N$, $N = 2$ (left). The rate of change of probability of a state (gray disk) is given by the divergence operator shown in the continuous case (middle right) and the discrete case (right).

The Applications

There’re a lot of cool applications!

Transfusion: Predict the Next Token and Diffuse Images with One Multi-Modal Model

Chunting Zhou[μ*] Lili Yu[μ*] Arun Babu[δ†] Kushal Tirumala[μ]
Michihiro Yasunaga[μ] Leonid Shamis[μ] Jacob Kahn[μ] Xuezhe Ma[σ]
Luke Zettlemoyer[μ] Omer Levy[†]

[μ] Meta
[δ] Waymo [σ] University of Southern California

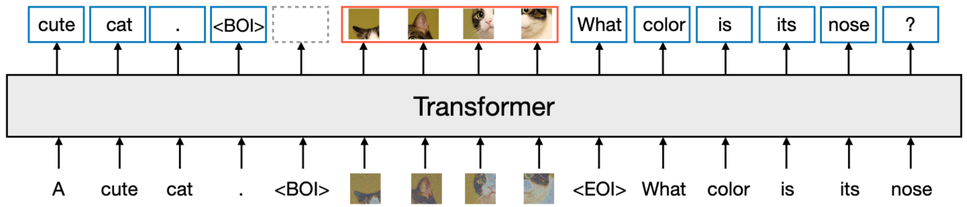
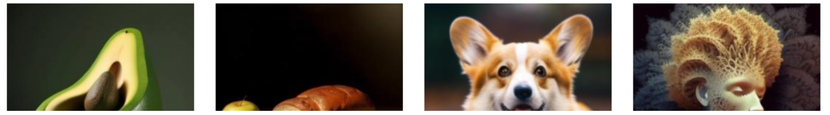


Figure 1: A high-level illustration of Transfusion. A single transformer perceives, processes, and produces data of every modality. Discrete (text) tokens are processed autoregressively and trained on the next token prediction objective. Continuous (image) vectors are processed together in parallel and trained on the diffusion objective. Marker BOI and EOI tokens separate the modalities.



FLUID: SCALING AUTOREGRESSIVE TEXT-TO-IMAGE GENERATIVE MODELS WITH CONTINUOUS TOKENS

Lijie Fan[1,*] Tianhong Li[2,†] Siyang Qin[1,†] Yuanzhen Li[1] Chen Sun[1]
Michael Rubinstein[1] Deqing Sun[1] Kaiming He[2] Yonglong Tian[1,*]
[1]Google DeepMind [2]MIT [*] equal contribution, project lead [†] equal contribution

Figure 1: Samples from our Fluid 10.5B autoregressive model with continuous tokens.

silly songs.

The Applications

There’re a lot of cool applications!

Meta

Movie Gen: A Cast of Media Foundation Models

The Movie Gen team @ Meta[1]
[1]A detailed contributor list can be found in the appendix of this paper.

We present MOVIE GEN, a cast of foundation models that generates high-quality, 1080p HD videos with different aspect ratios and synchronized audio. We also show additional capabilities such as precise instruction-based video editing and generation of personalized videos based on a user’s image. Our models set a new state-of-the-art on multiple tasks: text-to-video synthesis, video personalization, video editing, video-to-audio generation, and text-to-audio generation. Our largest video generation model is a 30B parameter transformer trained with a maximum context length of 73K video tokens, corresponding to a generated video of 16 seconds at 16 frames-per-second. We show multiple technical innovations and simplifications on the architecture, latent spaces, training objectives and recipes, data curation, evaluation protocols, parallelization techniques, and inference optimizations that allow us to reap the benefits of scaling pre-training data, model size, and training compute for training large scale media generation models. We hope this paper helps the research community to accelerate progress and innovation in media generation models.
All videos from this paper are available at https://go.fb.me/MovieGenResearchVideos.

Date: October 4, 2024
Blogpost: https://ai.meta.com/blog/movie-gen-media-foundation-models-generative-ai-video/

https://ai.meta.com/research/movie-gen/

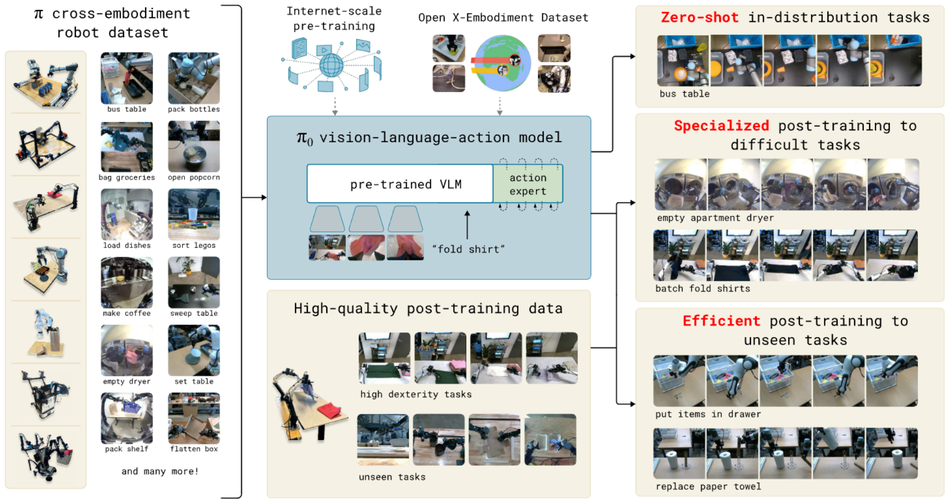
Figure 1 Examples of the different capabilities of Movie Gen. The MOVIE GEN cast of models generates videos from text prompts, supports the generation of videos consistent with characters in provided reference images, supports precise video editing given user provided instructions, and generates videos with synchronized audio. Videos in this Figure found at https://go.fb.me/MovieGen-Figure1.

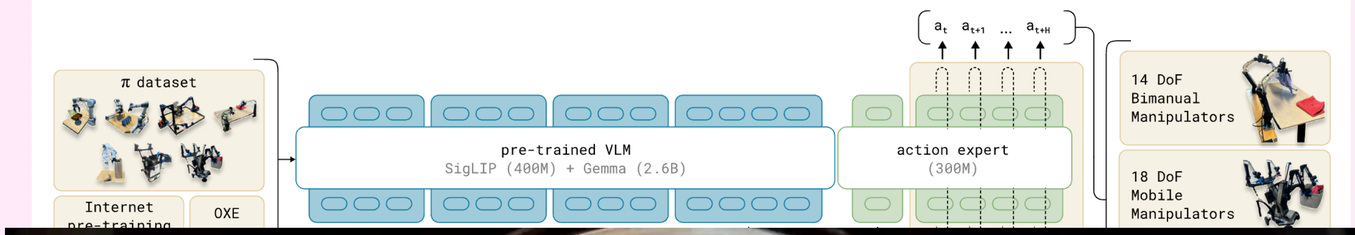
The Applications

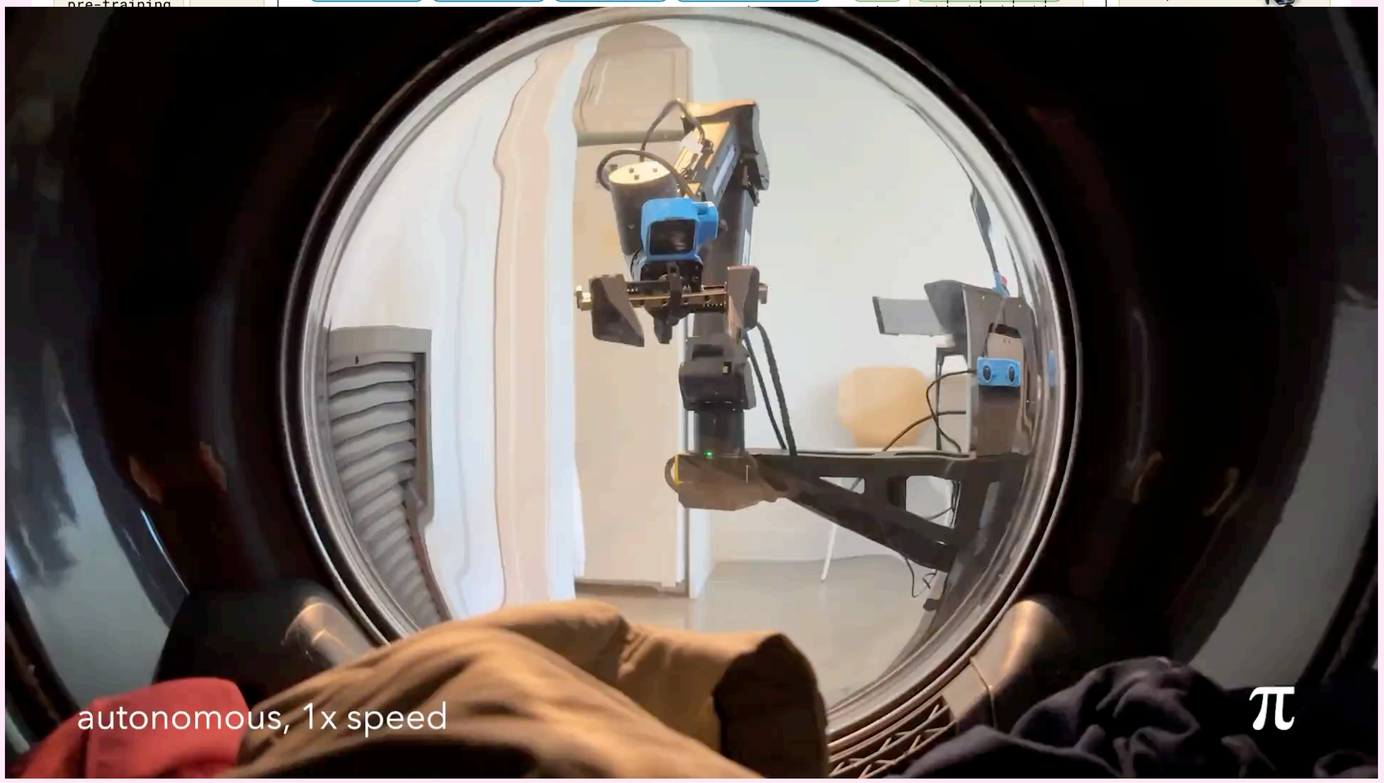
There’re a lot of cool applications!

π_0: A Vision-Language-Action Flow Model for General Robot Control

Physical Intelligence
Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, Ury Zhilinsky
https://physicalintelligence.company/blog/pi0



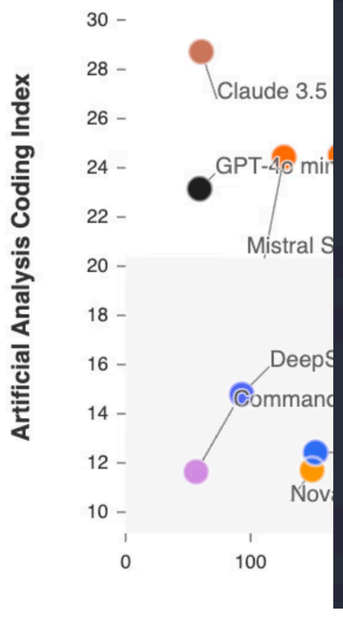


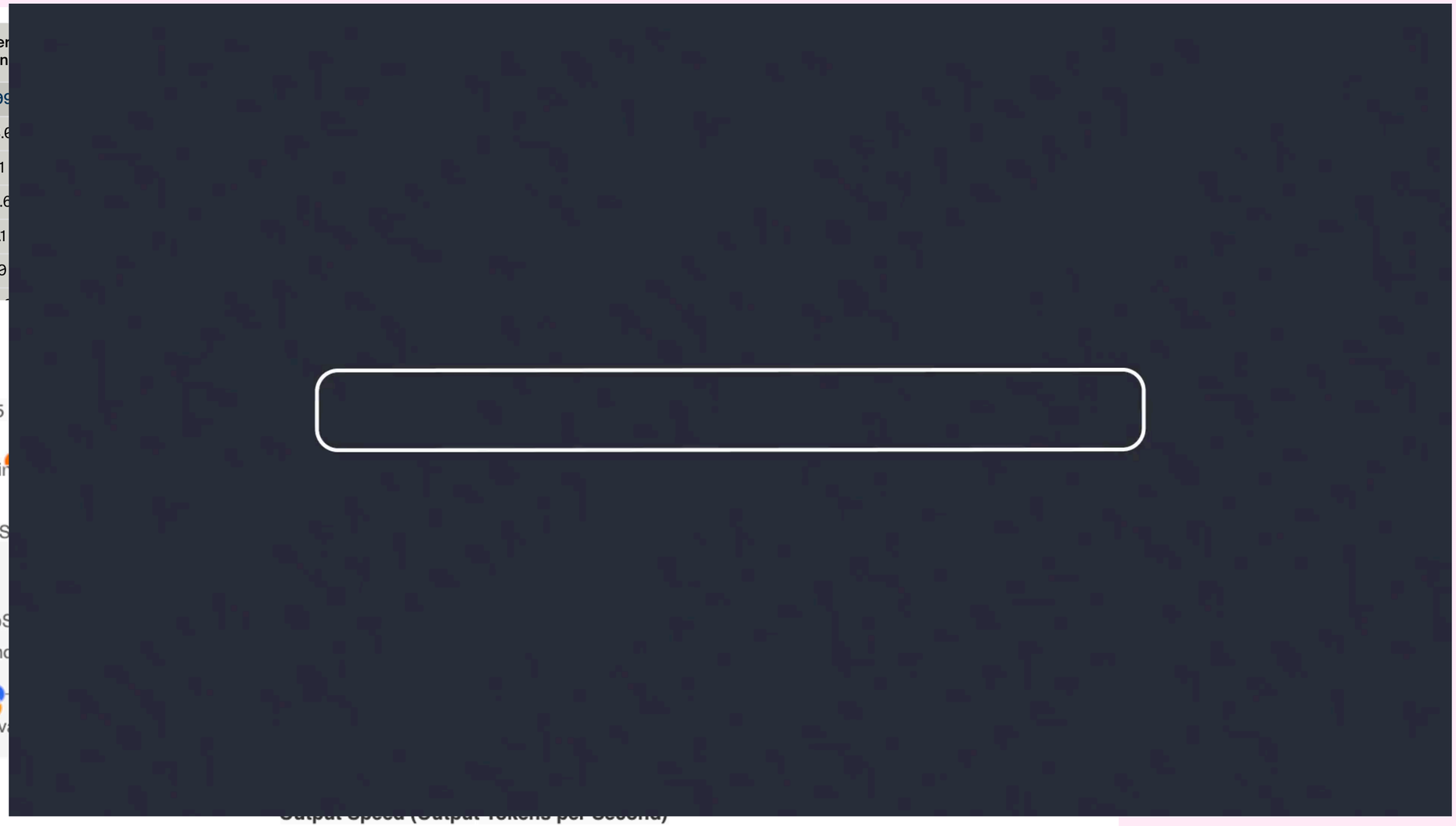


The Applications

There’re a lot of cool applications!

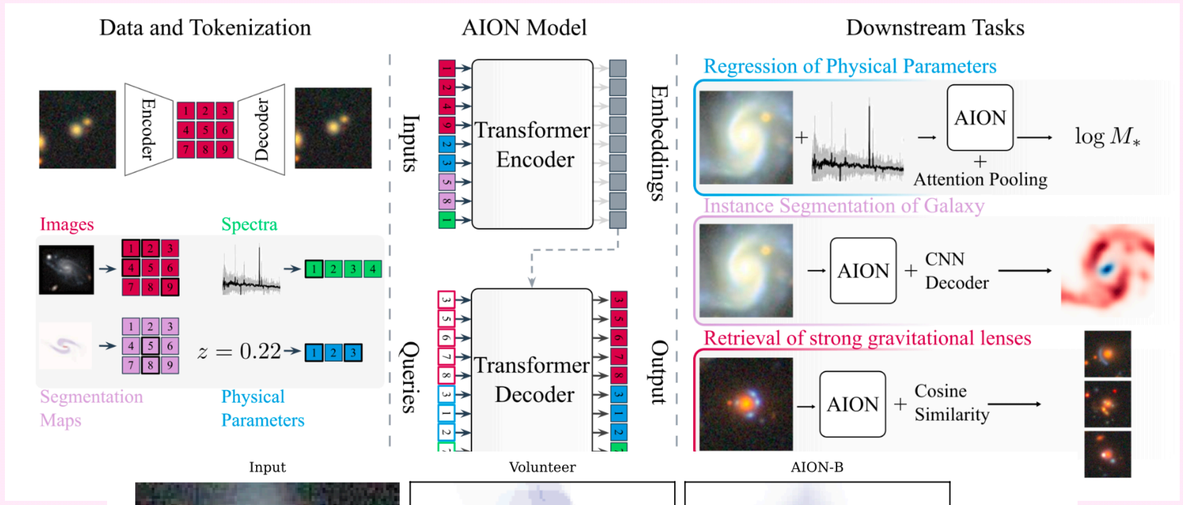
	Mer Min
Throughput (toks/sec)	1109
HumanEval	88.6
MBPP	77.1
EvalPlus	78.6
MultiPL-E	74.1
LiveCodeBench	17.0





The Applications

There’re a lot of cool applications!



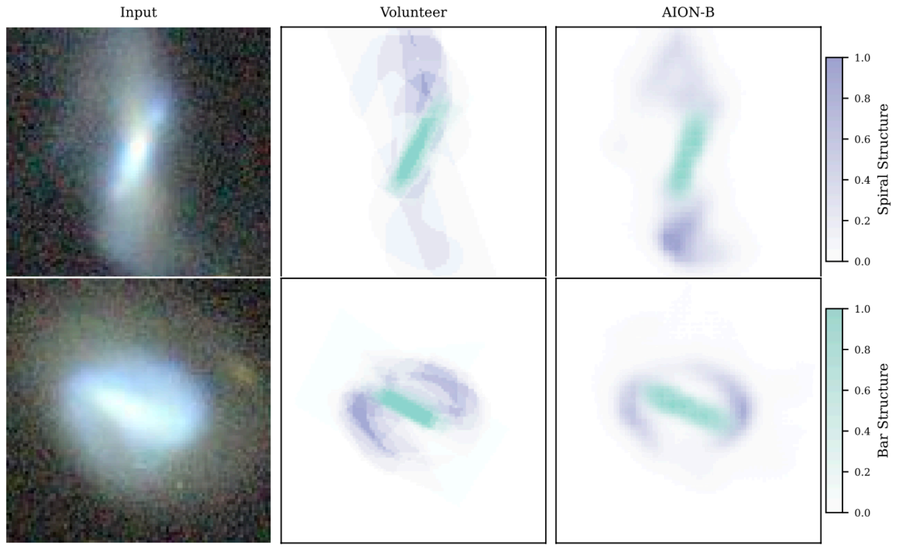
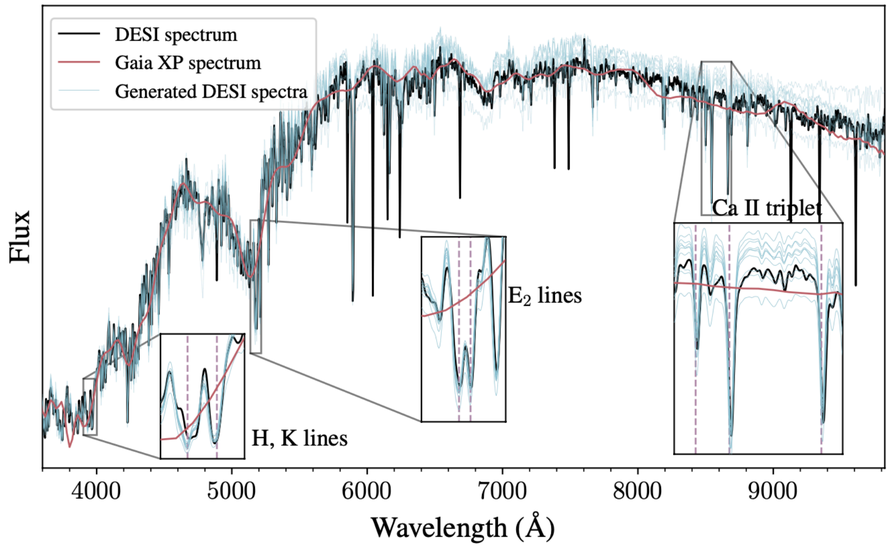


Figure 13: Exemplar ground truth image, crowd-annotated segmentation map, and AION-1 predictions.



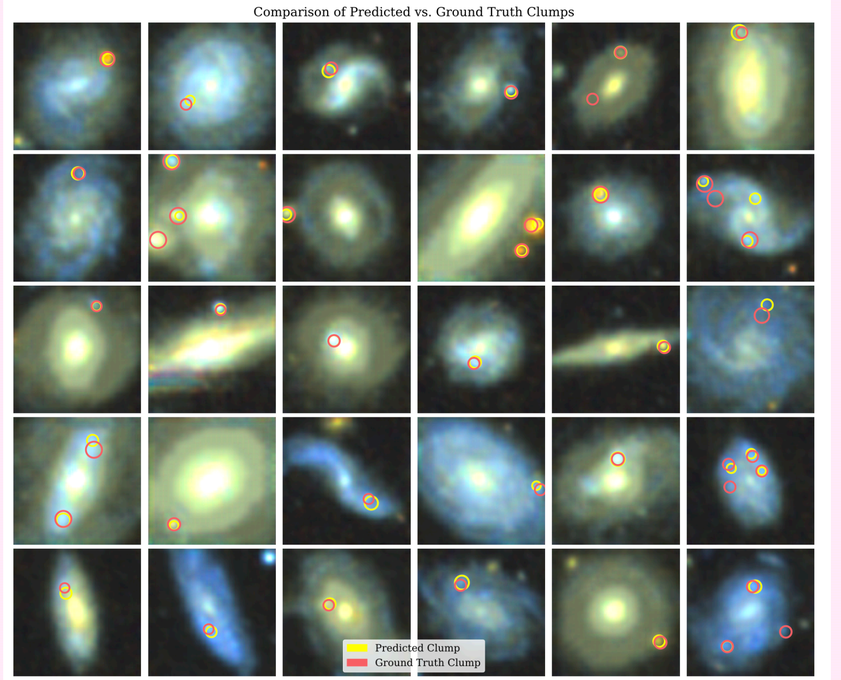


Figure 12: Qualitative examples of ground truth and predicted clump objects in Legacy Survey.

Thank You!