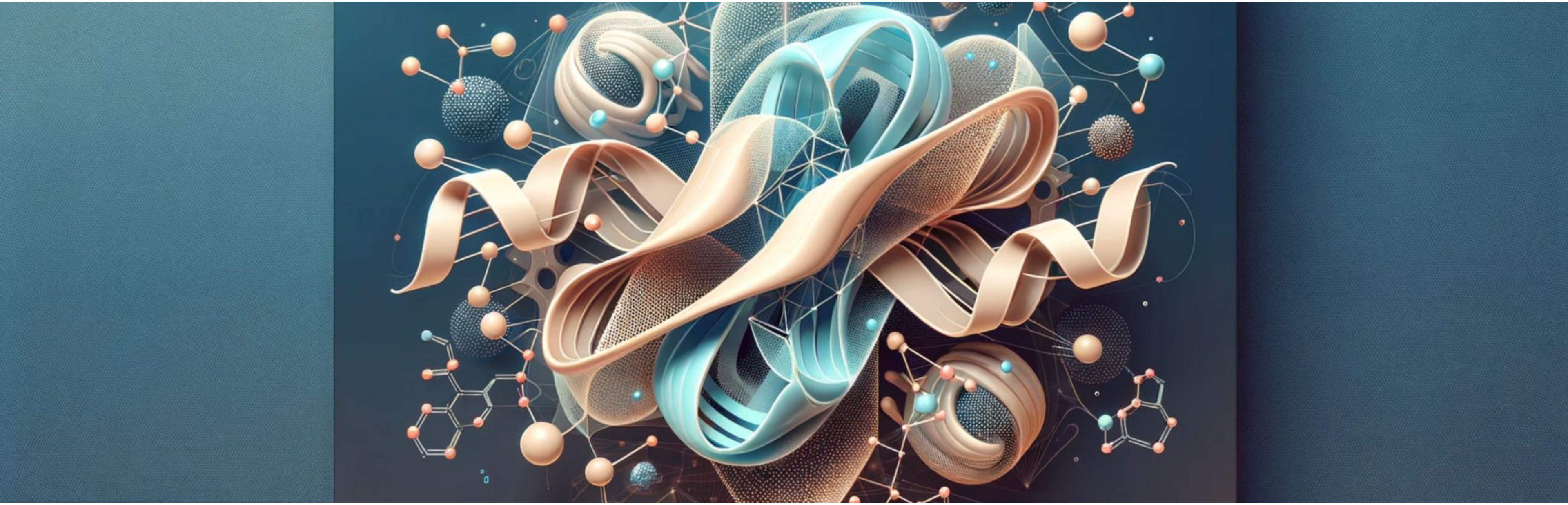
CSE7850/CX4803 Machine Learning in Computational Biology



Lecture 21: Programmable Protein Design

Yunan Luo

Controllable protein Design

- **Inverse folding**
  - Structure -> Sequence

Protein structure

Algorithm / Model

Amino acid sequence

MEKVNFLKNGVLRLPPGFRFRPTDEELVVQYLKRKVFSFPLPASIIPEVEVYKSDPWDLPGDMEQEKYFFSTK
EVKYPNGNRSNRATNSGYWKATGIDKQIILRGRQQQQQLIGLKKTLVFYRGKSPHGCRTNWIMHEYRLAN
LESNYHPIQGNWVICRIFLKKRGNTKNKEENMTTHDEVRNREIDKNSPVVSVKMSSRDSEALASANSELKK

Condition

Design

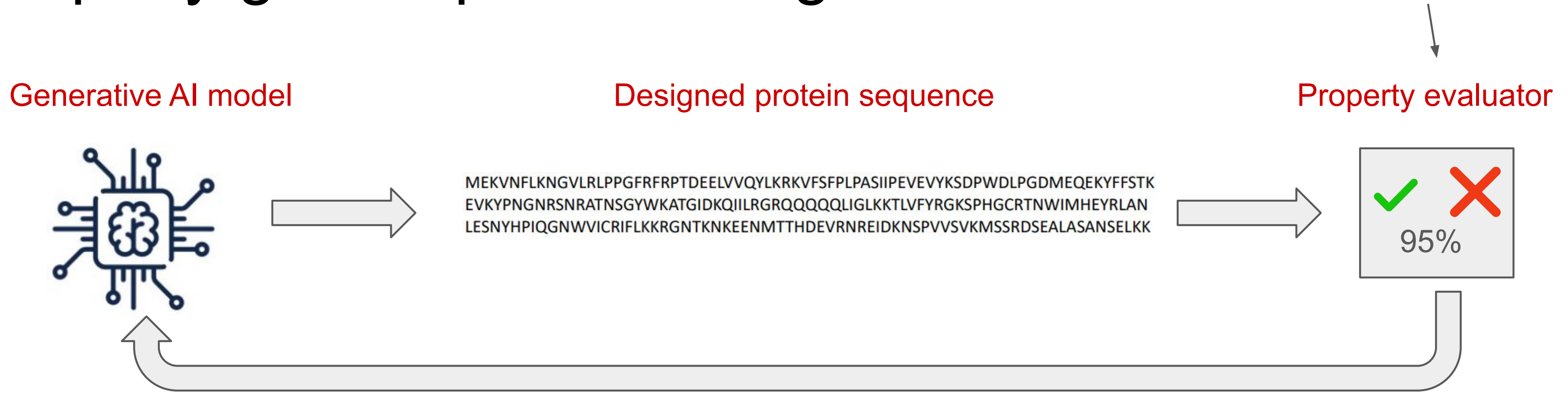
Sometimes, our goal could not be easily specified by a 3D structure input (because we don’t know such structures)

Examples:

- Generate a protein whose 99% residues are in alpha-helix
    - Or 80% residues, 50% residues, ..
- Generate a protein that can exhibit green fluorescence when exposed to light in the blue to ultraviolet range
- Generate a antibody protein that can bind to SARS-CoV-2 virus

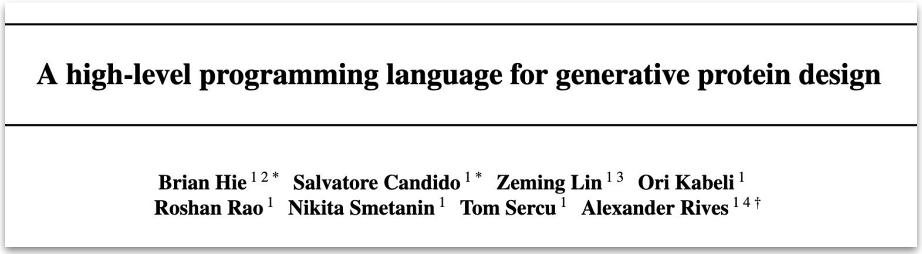
Property-guided protein design

Gold standard is wet-lab experiment, but often use computational model for design efficiency


Generative AI model
Designed protein sequence
Property evaluator
MEKVNFLKNGVLRLPPGFRFRPTDEELVVQYLKRKVFSFPLPASIIPEVEVYKSDPWDLPGDMEQEKYFFSTK
EVKYPNGNRSNRATNSGYWKATGIDKQIILRGRQQQQQLIGLKKTLVFYRGKSPHGCRTNWIMHEYRLAN
LESNYHPIQGNWVICRIFLKKRGNTKNKEENMTTHDEVRNREIDKNSPVVSVKMSSRDSEALASANSELKK
95%

- Generate a protein whose 99% residues are in alpha-helix
  - Count the residues
- Generate a protein that can exhibit green fluorescence when exposed to light in the blue to ultraviolet range
  - An ML model to predict the fluorescence brightness
- Generate a antibody protein that can bind to SARS-CoV-2 virus
  - An ML model to predict the binding affinity

Today's papers

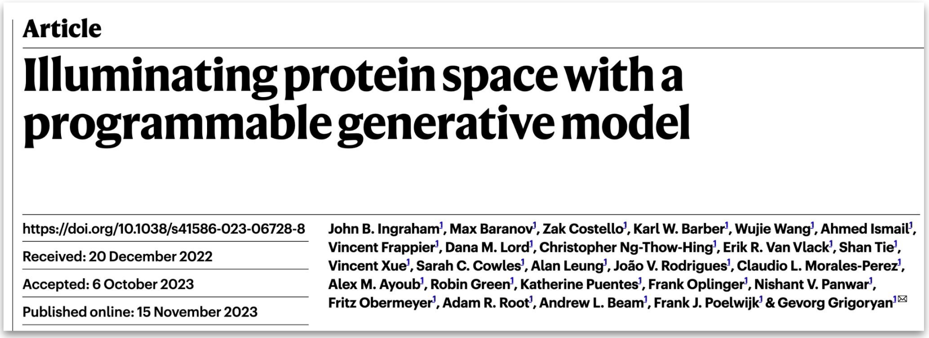


A high-level programming language for generative protein design

Brian Hie[1 2 *] Salvatore Candido[1 *] Zeming Lin[1 3] Ori Kabeli[1]
Roshan Rao[1] Nikita Smetanin[1] Tom Sercu[1] Alexander Rives[1 4 †]

Paper #1:

Use gradient-free scoring function to guide the design



Article

Illuminating protein space with a programmable generative model

https://doi.org/10.1038/s41586-023-06728-8

Received: 20 December 2022

Accepted: 6 October 2023

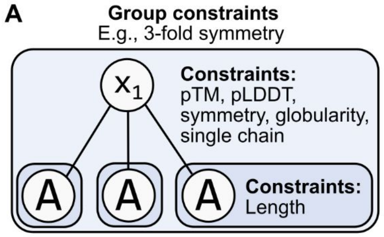
Published online: 15 November 2023

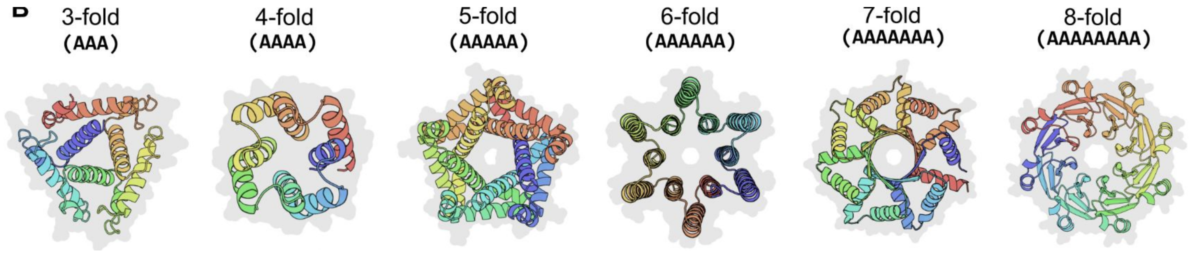
John B. Ingraham[1], Max Baranov[1], Zak Costello[1], Karl W. Barber[1], Wujie Wang[1], Ahmed Ismail[1], Vincent Frappier[1], Dana M. Lord[1], Christopher Ng-Thow-Hing[1], Erik R. Van Vlack[1], Shan Tie[1], Vincent Xue[1], Sarah C. Cowles[1], Alan Leung[1], João V. Rodrigues[1], Claudio L. Morales-Perez[1], Alex M. Ayoub[1], Robin Green[1], Katherine Puentes[1], Frank Oplinger[1], Nishant V. Panwar[1], Fritz Obermeyer[1], Adam R. Root[1], Andrew L. Beam[1], Frank J. Poelwijk[1] & Gevorg Grigoryan[1 ✉]

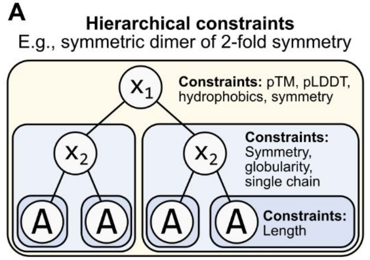
Paper #2:

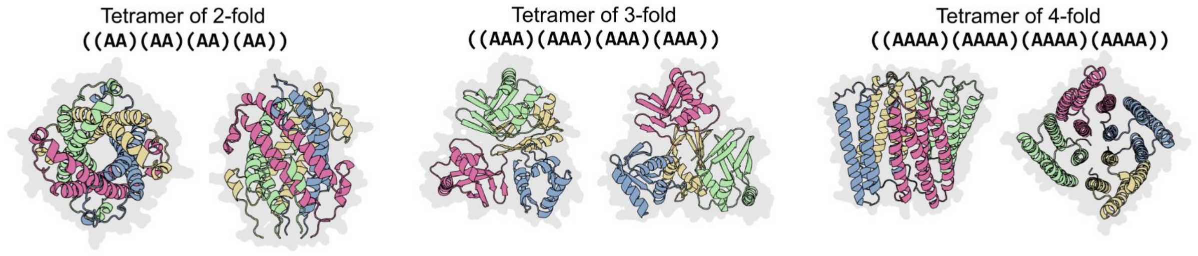
Use property evaluator (ML models) for which we can compute gradients to guide the design

Paper #1


A
Group constraints
E.g., 3-fold symmetry
x_1
Constraints: pTM, pLDDT, symmetry, globularity, single chain
A
A
A
Constraints: Length

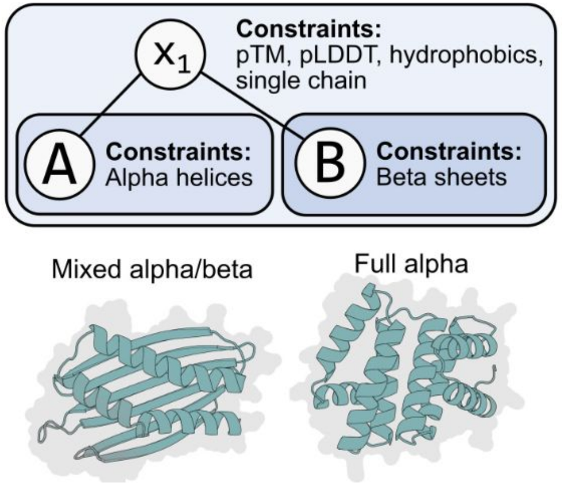

B
3-fold
(AAA)
4-fold
(AAAA)
5-fold
(AAAAA)
6-fold
(AAAAAA)
7-fold
(AAAAAAA)
8-fold
(AAAAAAAA)


A
Hierarchical constraints
E.g., symmetric dimer of 2-fold symmetry
x_1
Constraints: pTM, pLDDT, hydrophobics, symmetry
x_2
x_2
Constraints: Symmetry, globularity, single chain
A
A
A
A
Constraints: Length


Tetramer of 2-fold
((AA)(AA)(AA)(AA))
Tetramer of 3-fold
((AAA)(AAA)(AAA)(AAA))
Tetramer of 4-fold
((AAAA)(AAAA)(AAAA)(AAAA))

Paper #1

Goal: generate a alpha-helix protein

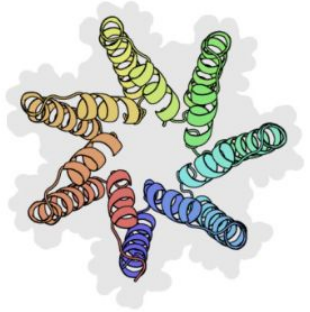


%(alpha)

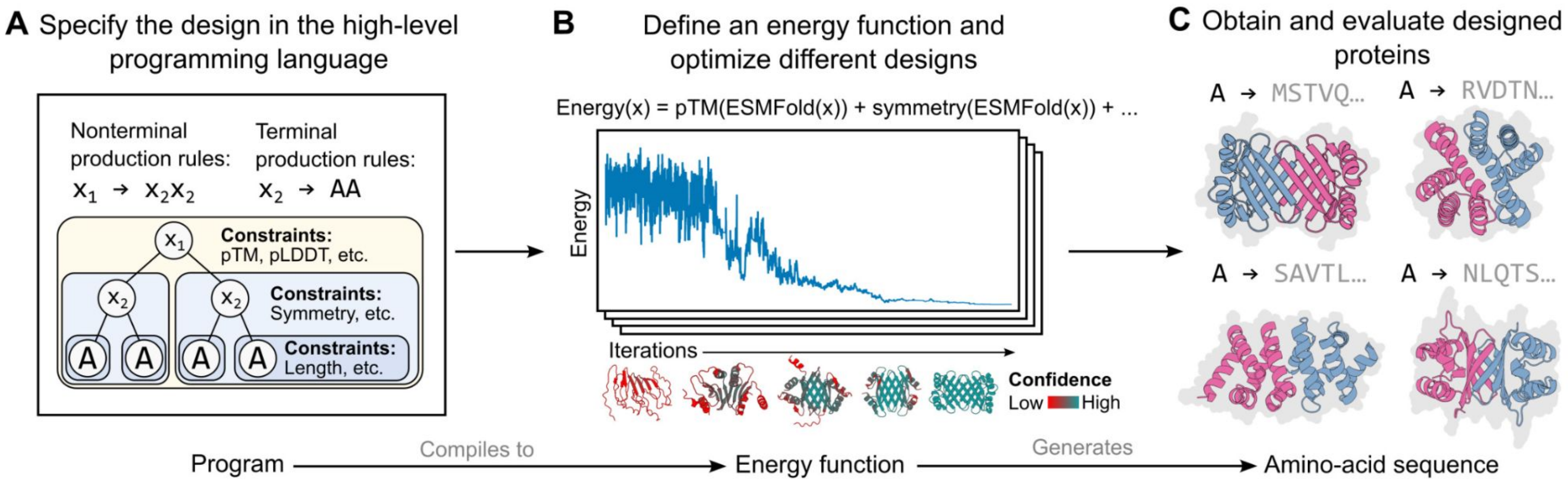
$$p_\theta(\mathbf{x}) = \frac{\exp(\boxed{-E_\theta(\mathbf{x})})}{Z(\theta)}$$

Sequence with high contents of alpha-helix will be sampled high probability

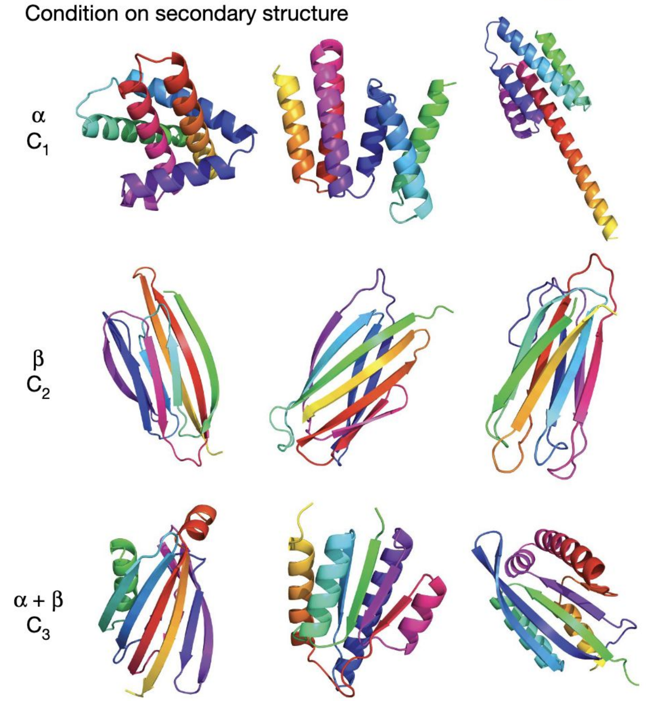
Question: how to define E(x) if we want to design a symmetric protein?

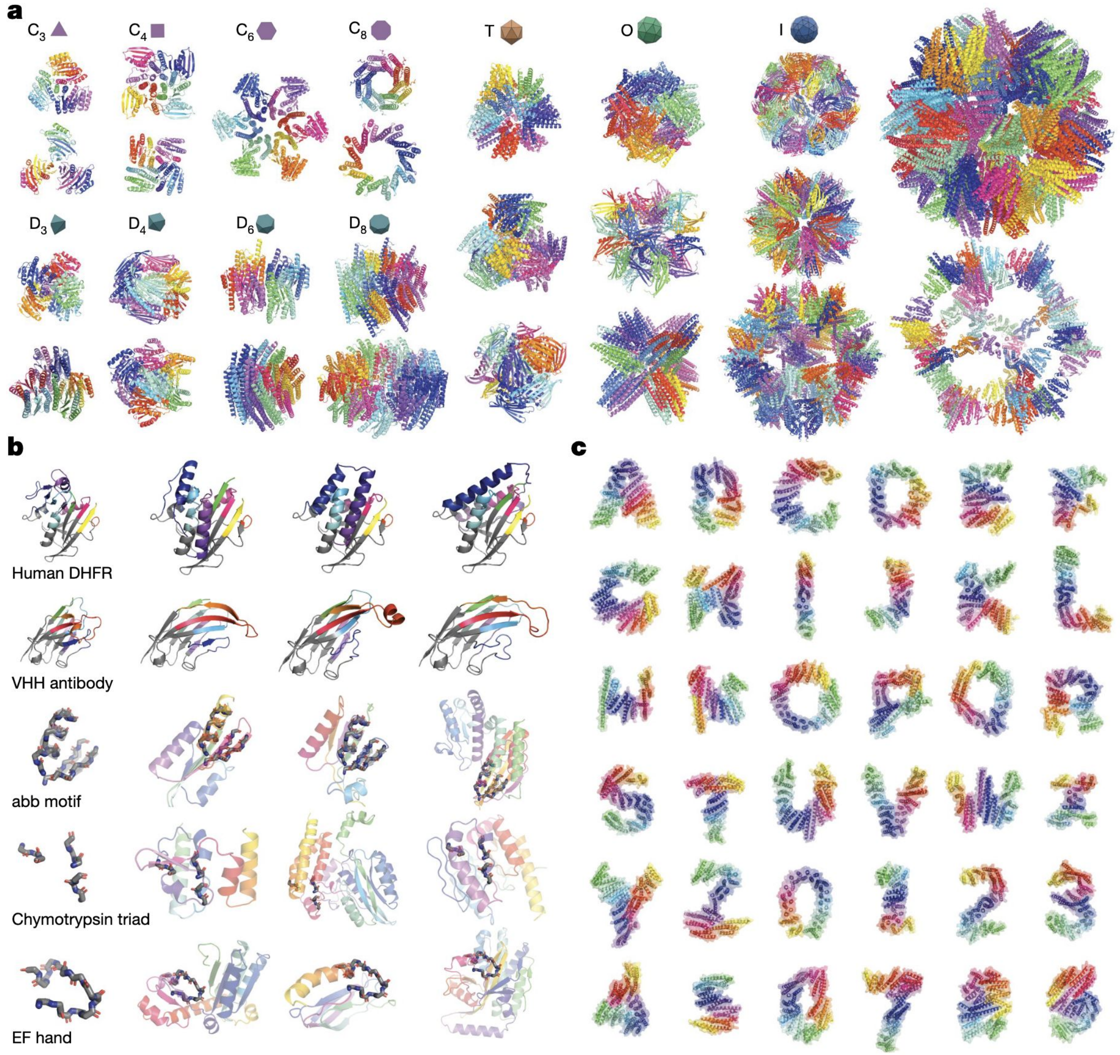


Paper #1

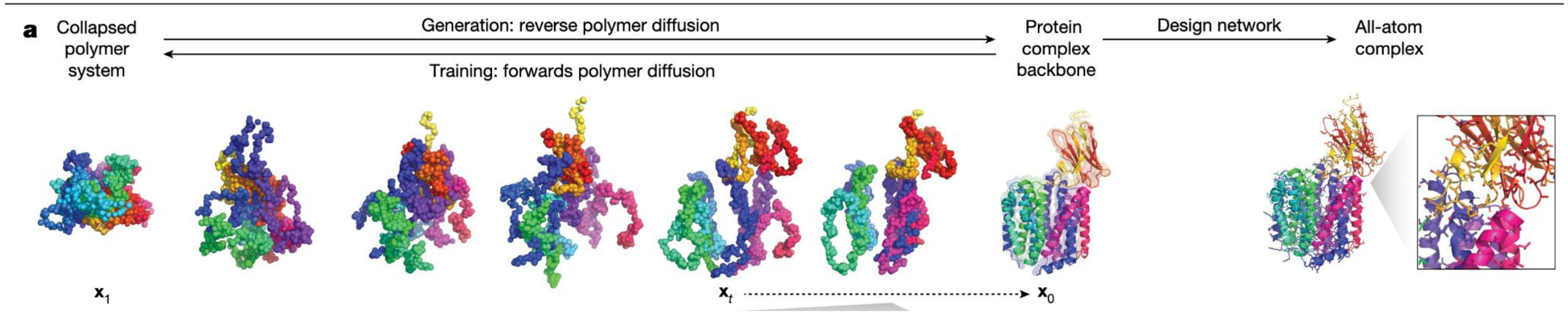

A Specify the design in the high-level programming language
Nonterminal production rules:
$x_1 \rightarrow x_2 x_2$
Terminal production rules:
$x_2 \rightarrow AA$
Constraints: pTM, pLDDT, etc.
Constraints: Symmetry, etc.
Constraints: Length, etc.
B Define an energy function and optimize different designs
Energy(x) = pTM(ESMFold(x)) + symmetry(ESMFold(x)) + ...
Energy
Iterations
Confidence
Low High
C Obtain and evaluate designed proteins
A → MSTVQ...
A → RVDTN...
A → SAVTL...
A → NLQTS...
Program
Compiles to
Energy function
Generates
Amino-acid sequence

Paper #2


Condition on secondary structure
α
C_1
β
C_2
α + β
C_3

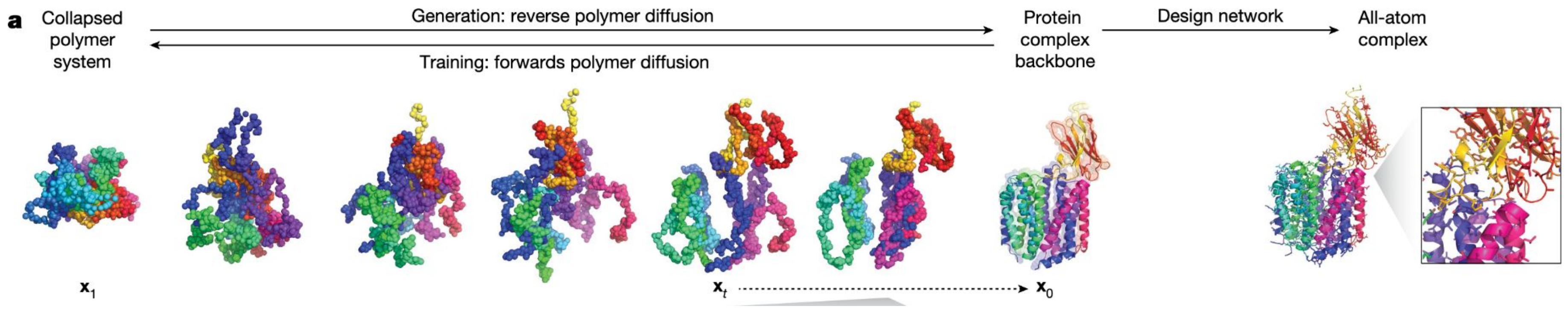

a
C_3
C_4
C_6
C_8
T
O
I
D_3
D_4
D_6
D_8
b
Human DHFR
VHH antibody
abb motif
Chymotrypsin triad
EF hand
c

Paper #2



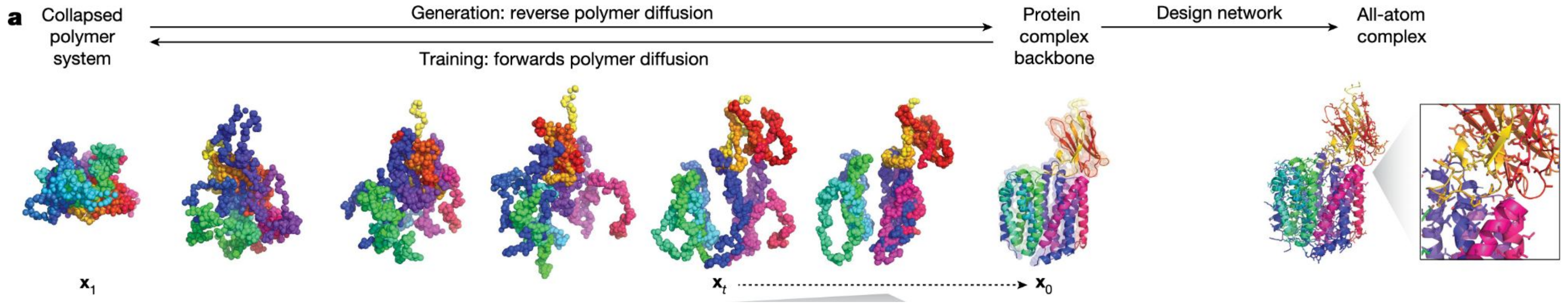
$$p(\text{protein})$$

Paper #2



$p(\text{protein}|\text{function})$ $p(\text{protein})$

Paper #2



$$\boxed{p(\text{protein}|\text{function})} \propto \boxed{p(\text{protein})} \times \boxed{p(\text{function}|\text{protein})}$$

Conditional generation

Unconditional generation

An ML predictor!

$$\begin{aligned}\nabla_{\mathbf{x}} \log p_t(\mathbf{x}|\mathbf{y}) &= \nabla_{\mathbf{x}} \log \frac{p_t(\mathbf{x})p_t(\mathbf{y}|\mathbf{x})}{p_t(\mathbf{y})} \\ &= \nabla_{\mathbf{x}} \log p_t(\mathbf{x}) + \nabla_{\mathbf{x}} \log p_t(\mathbf{y}|\mathbf{x}) - \cancel{\nabla_{\mathbf{x}} \log p_t(\mathbf{y})} \\ &= \nabla_{\mathbf{x}} \log \boxed{p_t(\mathbf{x})} + \nabla_{\mathbf{x}} \log \boxed{p_t(\mathbf{y}|\mathbf{x})}.\end{aligned}$$

Unconditional generative model

Property predictor

Diffusion Models

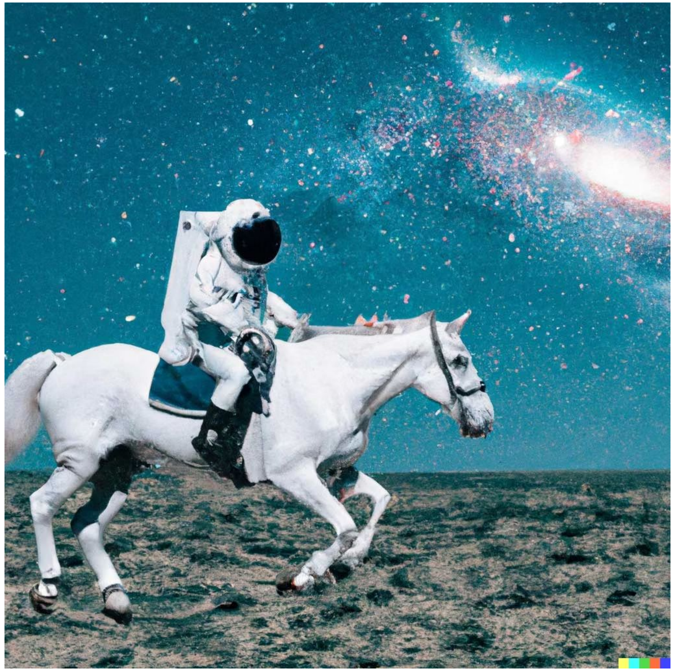
Ho et al. Denoising diffusion probabilistic models (DDPM), Neurips 2020.
Song et al. Score-based generative modeling through stochastic differential equations, ICLR 2021.
Bao et al. Analytic-DPM: an Analytic Estimate of the Optimal Reverse Variance in Diffusion Probabilistic Models, ICLR 2022.
Bao et al. Estimating the Optimal Covariance with Imperfect Mean in Diffusion Probabilistic Models, ICML 2022.
Rombach et al. High-resolution image synthesis with latent diffusion models. CVPR, 2022.

Text-to-image generation

Input

An astronaut riding a horse in photorealistic style.

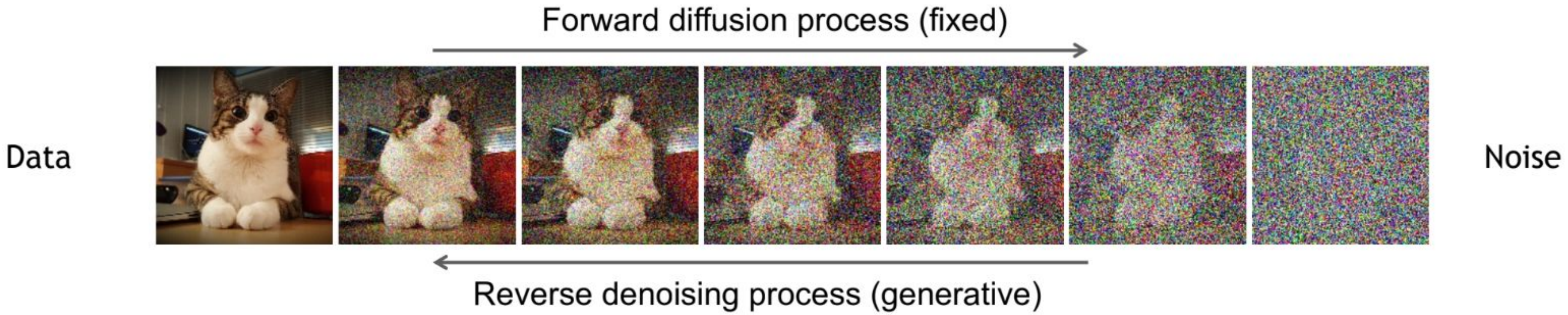
Output



Diffusion models

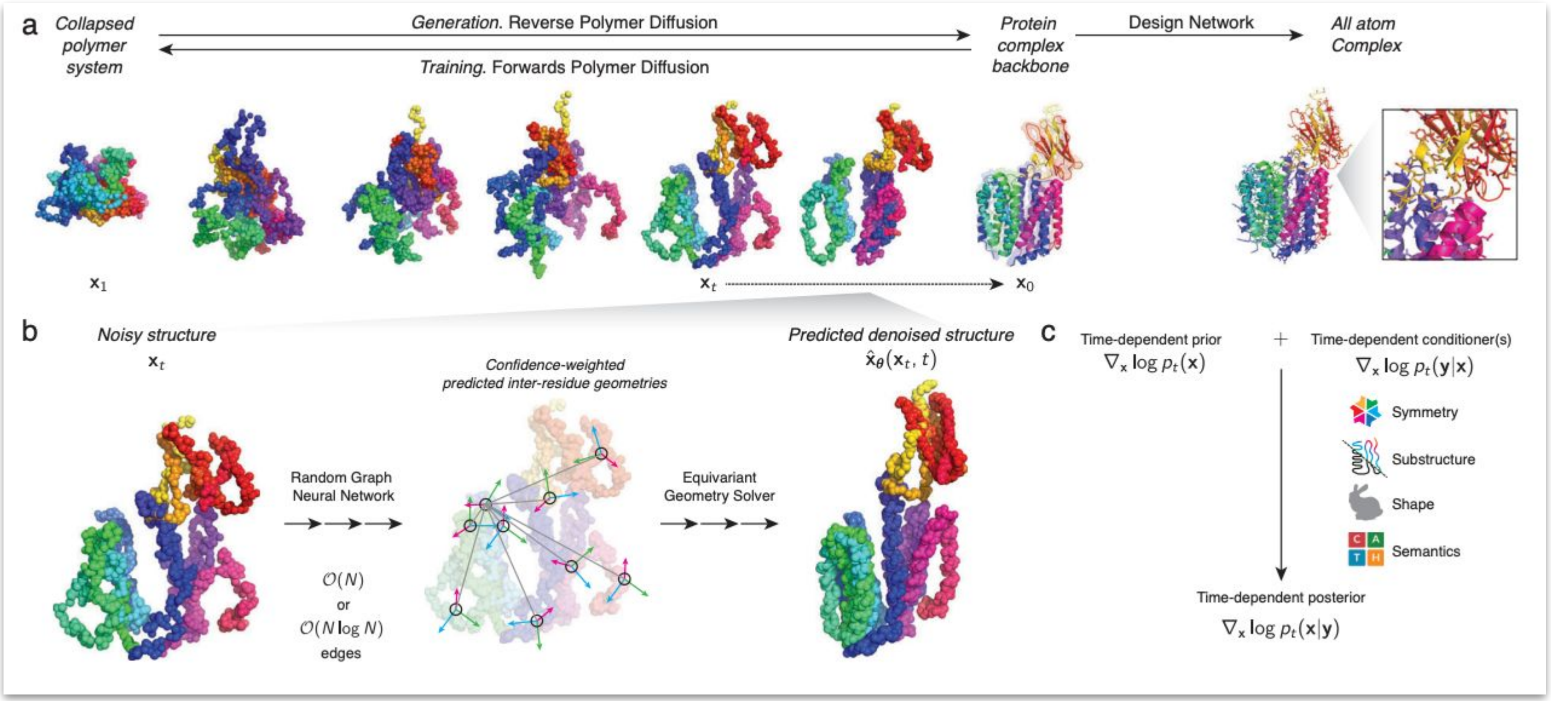
Denoising diffusion models consist of two processes:

- Forward diffusion process that gradually adds noise to input
- Reverse denoising process that learns to generate data by denoising


Forward diffusion process (fixed)
Data
Noise
Reverse denoising process (generative)

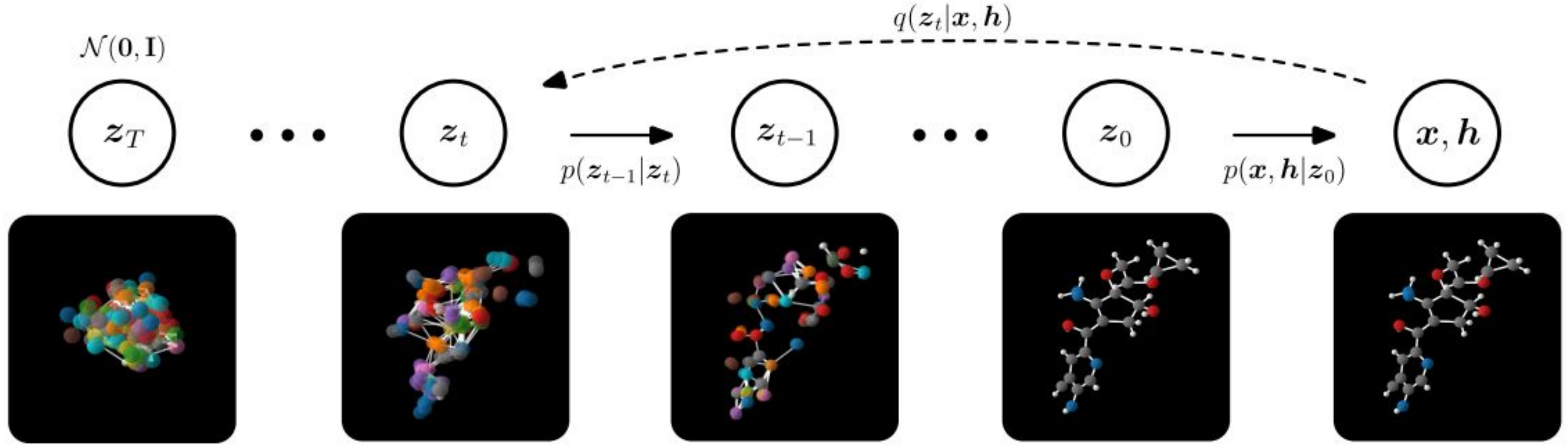
Slides credits: https://cvpr2022-tutorial-diffusion-models.github.io/

Application: Protein Design


a
Collapsed polymer system
Generation. Reverse Polymer Diffusion
Training. Forwards Polymer Diffusion
Protein complex backbone
Design Network
All atom Complex
$\mathbf{x}_1$
$\mathbf{x}_t$
$\mathbf{x}_0$
b
Noisy structure
$\mathbf{x}_t$
Random Graph Neural Network
$\mathcal{O}(N)$
or
$\mathcal{O}(N \log N)$
edges
Confidence-weighted predicted inter-residue geometries
Equivariant Geometry Solver
Predicted denoised structure
$\hat{\mathbf{x}}_\theta(\mathbf{x}_t, t)$
c
Time-dependent prior
$\nabla_\mathbf{x} \log p_t(\mathbf{x})$
+
Time-dependent conditioner(s)
$\nabla_\mathbf{x} \log p_t(\mathbf{y}|\mathbf{x})$
Symmetry
Substructure
Shape
Semantics
Time-dependent posterior
$\nabla_\mathbf{x} \log p_t(\mathbf{x}|\mathbf{y})$

Ingraham et al., “Illuminating protein space with a programmable generative model”, bioRxiv, 2022

Application: Drug Design



$\mathcal{N}(\mathbf{0}, \mathbf{I})$
$\boldsymbol{z}_T$
$\boldsymbol{z}_t$
$\boldsymbol{z}_{t-1}$
$\boldsymbol{z}_0$
$\boldsymbol{x}, \boldsymbol{h}$
$p(\boldsymbol{z}_{t-1} | \boldsymbol{z}_t)$
$p(\boldsymbol{x}, \boldsymbol{h} | \boldsymbol{z}_0)$
$q(\boldsymbol{z}_t | \boldsymbol{x}, \boldsymbol{h})$

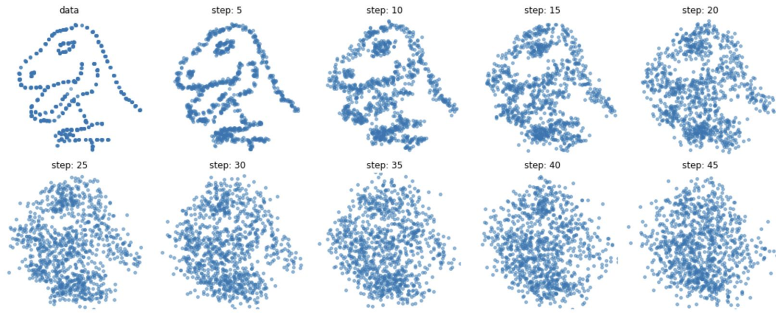
Hoogeboom et al., “Equivariant Diffusion for Molecule Generation in 3D”, ICML, 2022

Simple code demo

https://github.com/tanelp/tiny-diffusion

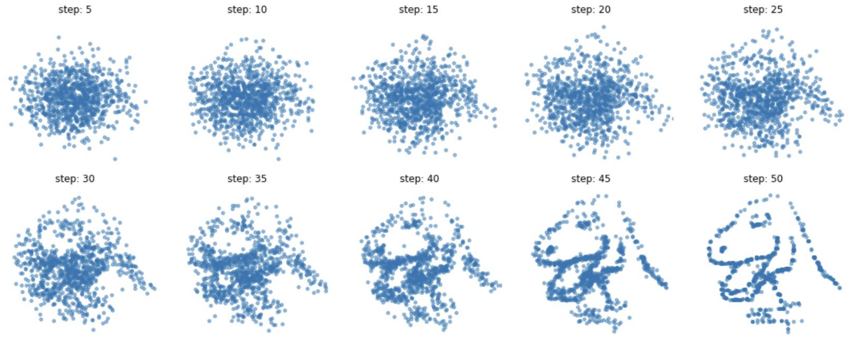
Forward process

A visualization of the forward diffusion process being applied to a dataset of one thousand 2D points. Note that the dinosaur is not a single training example, it represents each 2D point in the dataset.

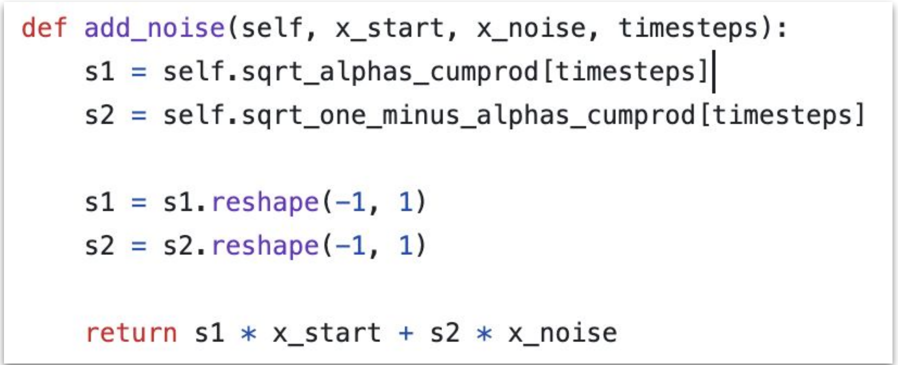


Reverse process

This illustration shows how the reverse process recovers the distribution of the training data.

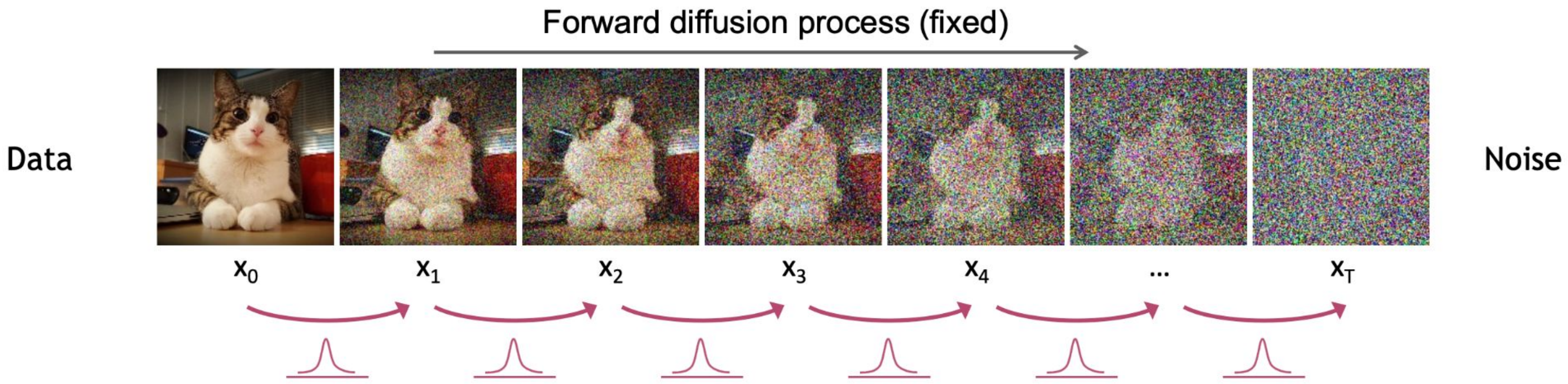


Forward Diffusion Process



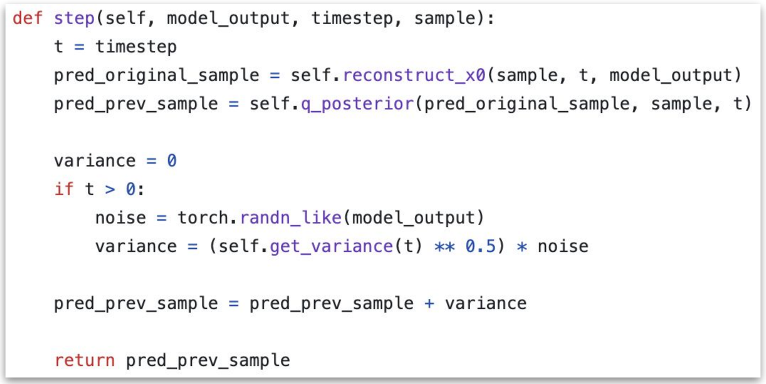
```
def add_noise(self, x_start, x_noise, timesteps):
    s1 = self.sqrt_alphas_cumprod[timesteps]
    s2 = self.sqrt_one_minus_alphas_cumprod[timesteps]

    s1 = s1.reshape(-1, 1)
    s2 = s2.reshape(-1, 1)

    return s1 * x_start + s2 * x_noise
```

The formal definition of the forward process in T steps:



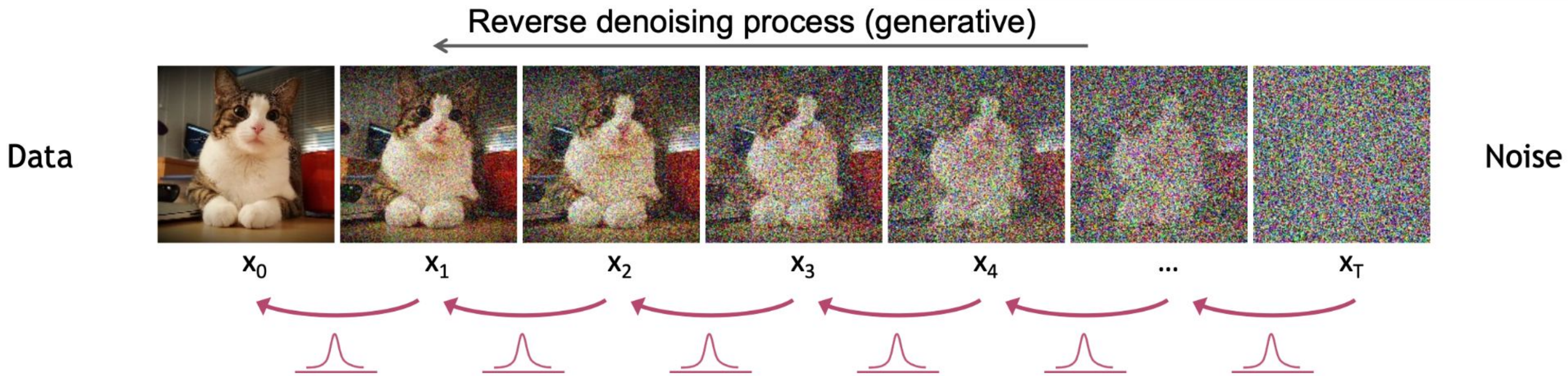
$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x_t}; \sqrt{1-\beta_t}\mathbf{x_{t-1}}, \beta_t\mathbf{I}) \quad \Rightarrow \quad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \qquad \text{(joint)}$$

Reverse Denoising Process



```
def step(self, model_output, timestep, sample):
    t = timestep
    pred_original_sample = self.reconstruct_x0(sample, t, model_output)
    pred_prev_sample = self.q_posterior(pred_original_sample, sample, t)

    variance = 0
    if t > 0:
        noise = torch.randn_like(model_output)
        variance = (self.get_variance(t) ** 0.5) * noise

    pred_prev_sample = pred_prev_sample + variance

    return pred_prev_sample
```

Formal definition of forward and reverse processes in T steps:



$$p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$$

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \underbrace{\mu_\theta(\mathbf{x}_t, t)}_{\substack{\text{Trainable network} \\ \text{(U-net, Denoising Autoencoder)}}}, \sigma_t^2\mathbf{I})$$

$$\Rightarrow \quad p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$$