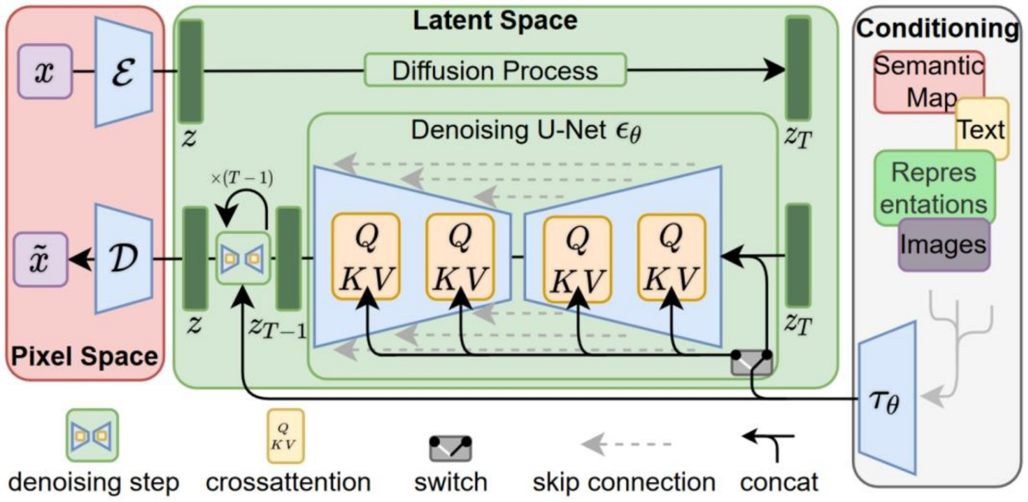


Coding Stable Diffusion form scratch in PyTorch

Umar Jamil

Topics and Prerequisites

Topics discussed

- Latent Diffusion Models (Stable Diffusion) from scratch in PyTorch. No other libraries used except for tokenizer.
- Maths of diffusion models as defined in the DDPM paper (simplified!)
- Classifier-Free Guidance
- Text – to – Image
- Image – to – Image
- Inpainting

Future videos

- Score-based models
- ODE and SDE theoretical framework for diffusion models
- Euler, Runge-Kutta and derived samplers.

Prerequisites

- Basics of probability and statistics (multivariate gaussian, conditional probability, marginal probability, likelihood, Bayers' rule).
    - I will give a non-maths intuition for most concepts.
- Basics of PyTorch and neural networks
- How the attention mechanism works (watch my video on the Transformer model).
- How convolution layers work

What is Stable Diffusion?

Stable Diffusion is a text-to-image deep learning model, based on diffusion models.
Introduced in 2022, developed by the CompViz Group at LMU Munich.
https://github.com/Stability-AI/stablediffusion

Prompt

Picture of a dog with glasses

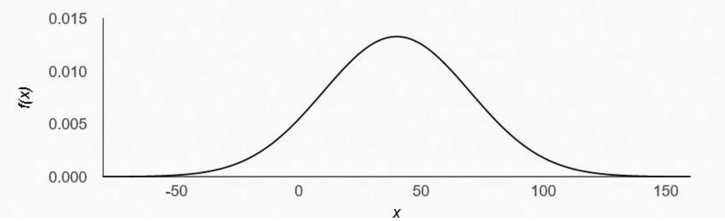
Output

What is a generative model?

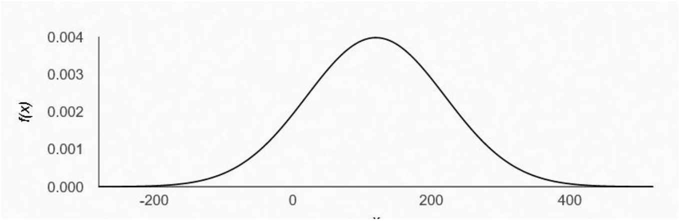
A generative model learns a probability distribution of the data set such that we can then sample from the distribution to create r instances of data. For example, if we have many pictures of cats and we train a generative model on it, we then sample from this distribution to create new images of cats.

Why do we model data as distributions?

- Imagine you’re a criminal, and you want to generate thousands of fake identities. Each fake identity, is made up of variables, representing the characteristics of a person (Age, Height).
- You can ask the Statistics Department of the Government to give you statistics about the age and the height of the population and then sample from these distributions.

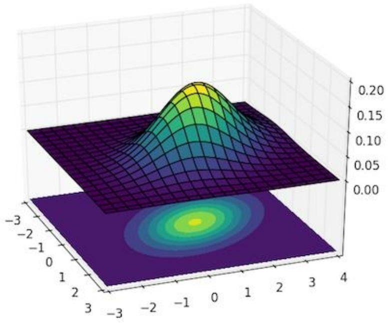


Age: **N**$(40, 30^2)$



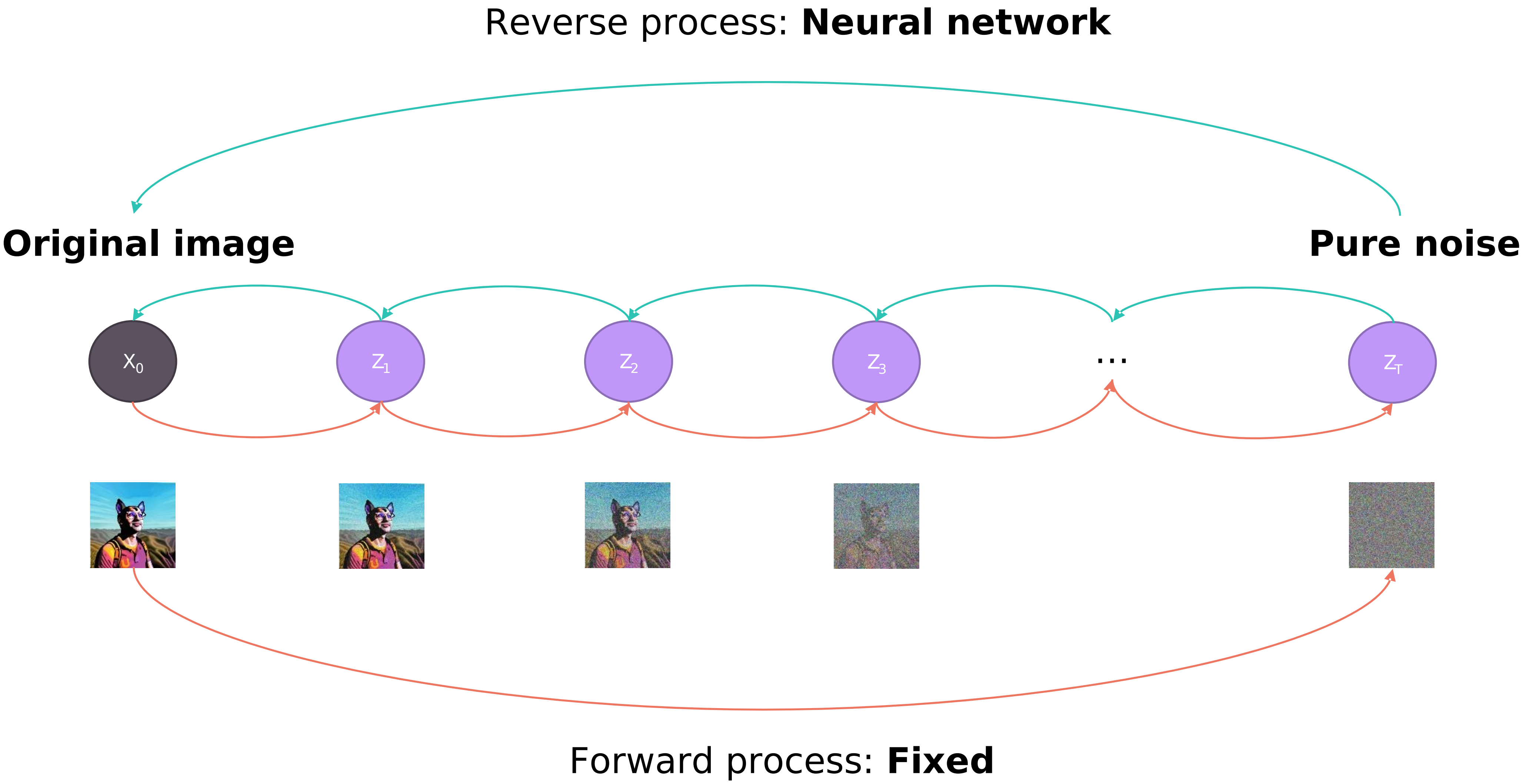
Height: **N**$(120, 100^2)$

- At first, you may sample from each distribution independently to cre fake identity, but that would produce unreasonable pairs of (Age, He
- To generate fake identities that make sense, you need the **joint distribution**, otherwise you may end up with an unreasonable pair (Age, Height)
- We can also evaluate probabilities on one of the two variables using **conditional probability** and/or by **marginalizing** a variable.



Learning the distribution $p(x)$ of our data.

- We have a data set made of up images, and we want to learn a very complex distribution that we can then use to **sample** from.



Reverse process: **Neural network**

Original image

Pure noise

Forward process: **Fixed**

The math of diffusion models… simplified

Just like with a VAE, we want to learn the parameters of the latent space

Reverse process **p**

2 Background

Diffusion models [53] are latent variable models of the form $p_\theta(\mathbf{x}_0) := \int p_\theta(\mathbf{x}_{0:T})\, d\mathbf{x}_{1:T}$, where $\mathbf{x}_1, \ldots, \mathbf{x}_T$ are latents of the same dimensionality as the data $\mathbf{x}_0 \sim q(\mathbf{x}_0)$. The joint distribution $p_\theta(\mathbf{x}_{0:T})$ is called the *reverse process*, and it is defined as a Markov chain with learned Gaussian transitions starting at $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$:

$$p_\theta(\mathbf{x}_{0:T}) := p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t), \qquad p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)) \tag{1}$$

What distinguishes diffusion models from other types of latent variable models is that the approximate posterior $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$, called the *forward process* or *diffusion process*, is fixed to a Markov chain that gradually adds Gaussian noise to the data according to a variance schedule $\beta_1, \ldots, \beta_T$:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) := \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}), \qquad q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \tag{2}$$

Evidence Lower Bound **(ELBO**

Training is performed by optimizing the usual variational bound on negative log likelihood:

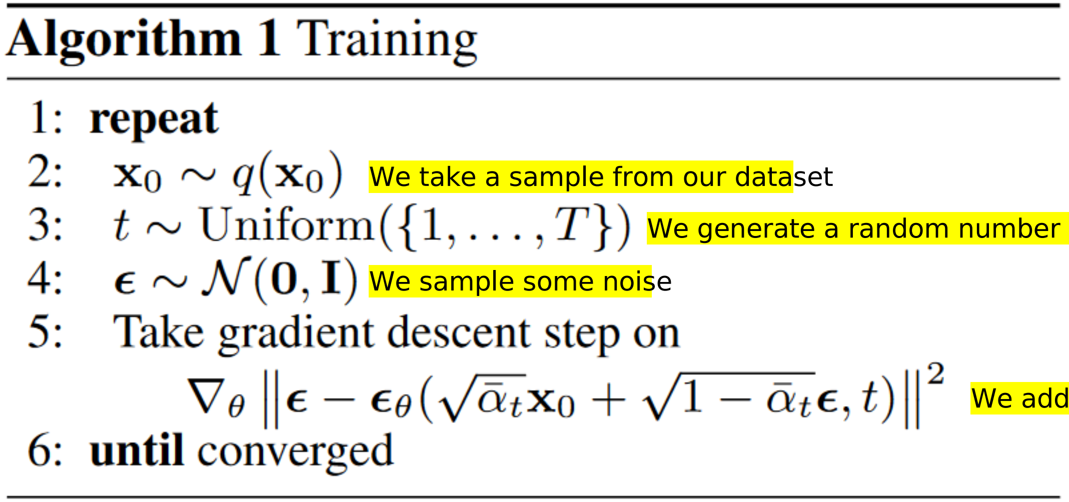
$$\mathbb{E}\left[-\log p_\theta(\mathbf{x}_0)\right] \le \mathbb{E}_q\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] = \mathbb{E}_q\left[-\log p(\mathbf{x}_T) - \sum_{t\ge 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})}\right] =: L \tag{3}$$

Forward process **q**

The forward process variances β_t can be learned by reparameterization [33] or held constant as hyperparameters, and expressiveness of the reverse process is ensured in part by the choice of Gaussian conditionals in $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$, because both processes have the same functional form when β_t are small [53]. A notable property of the forward process is that it admits sampling $\mathbf{x}_t$ at an arbitrary timestep t in closed form: using the notation $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$, we have

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{4}$$

Ho, J., Jain, A. and Abbeel, P., 2020. Denoising diffusion probabilistic models. Advances in Neural Information Processing Systems, 33, pp.6840-6851.



Algorithm 1 Training

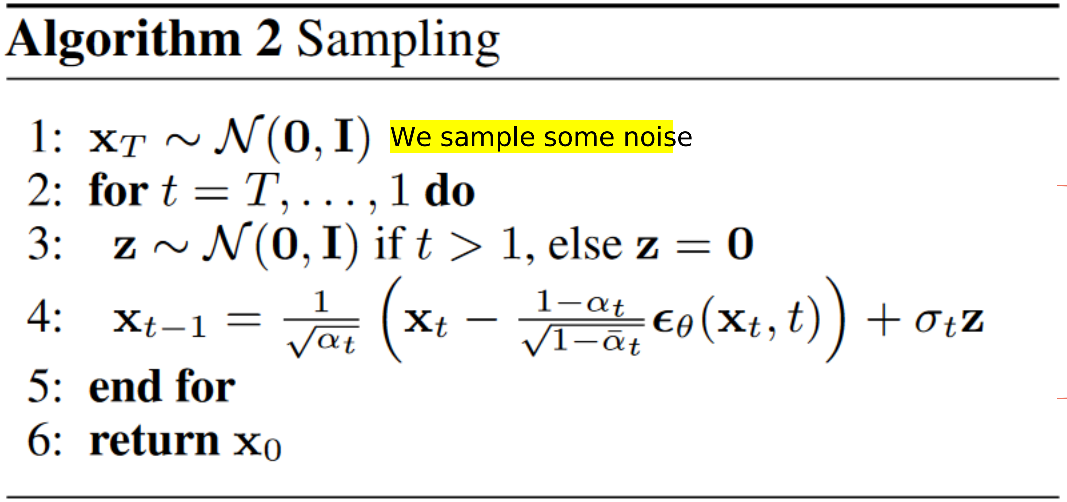
```
repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient descent step on
      $\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$
until converged
```

We take a sample from our dataset

We generate a random number t, between 1 and T

We sample some noise

We add noise to our image, and we train the model to learn to predict the amount of noise present in it.



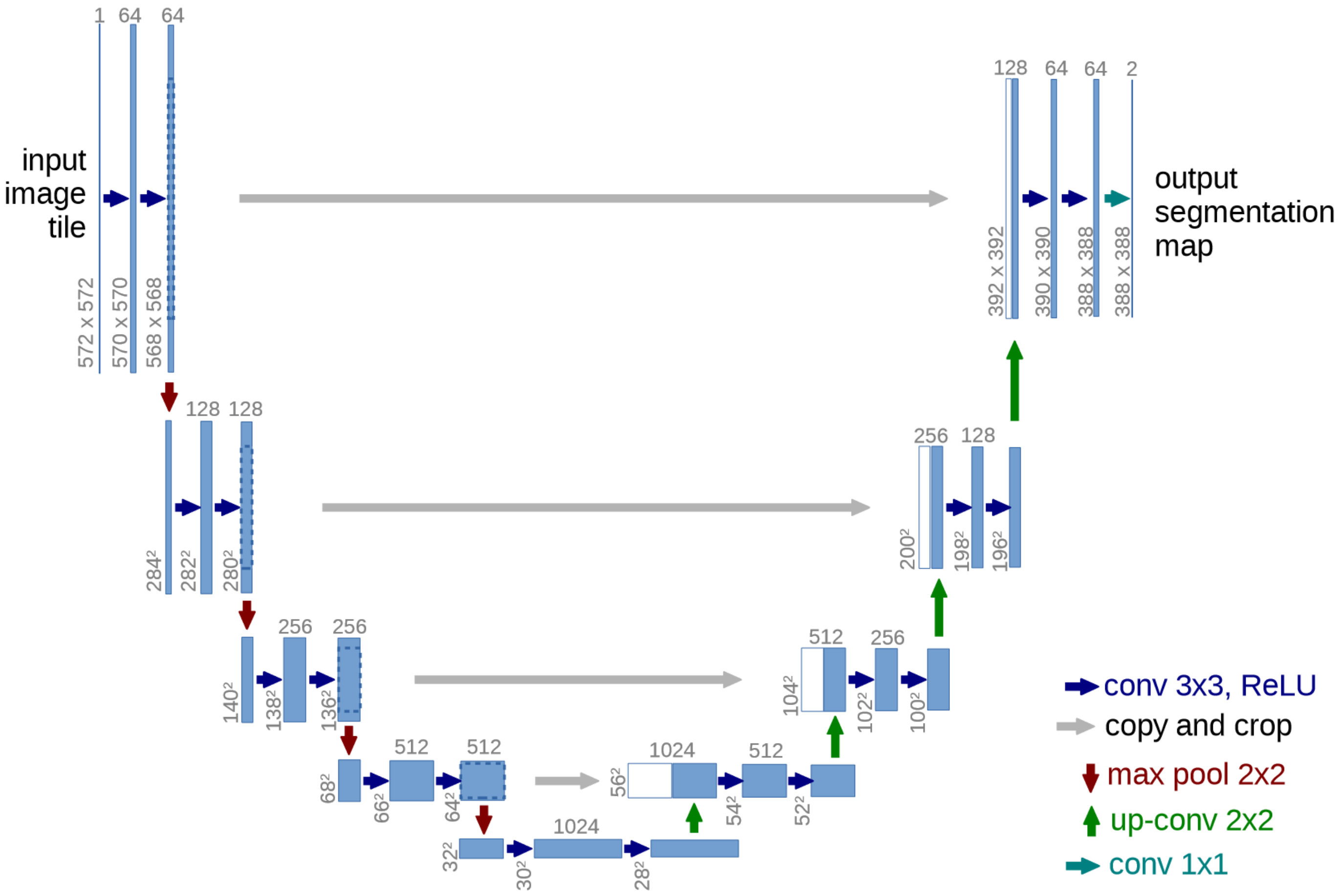
Algorithm 2 Sampling

```
$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \ldots, 1$ do
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
end for
return $\mathbf{x}_0$
```

We sample some noise

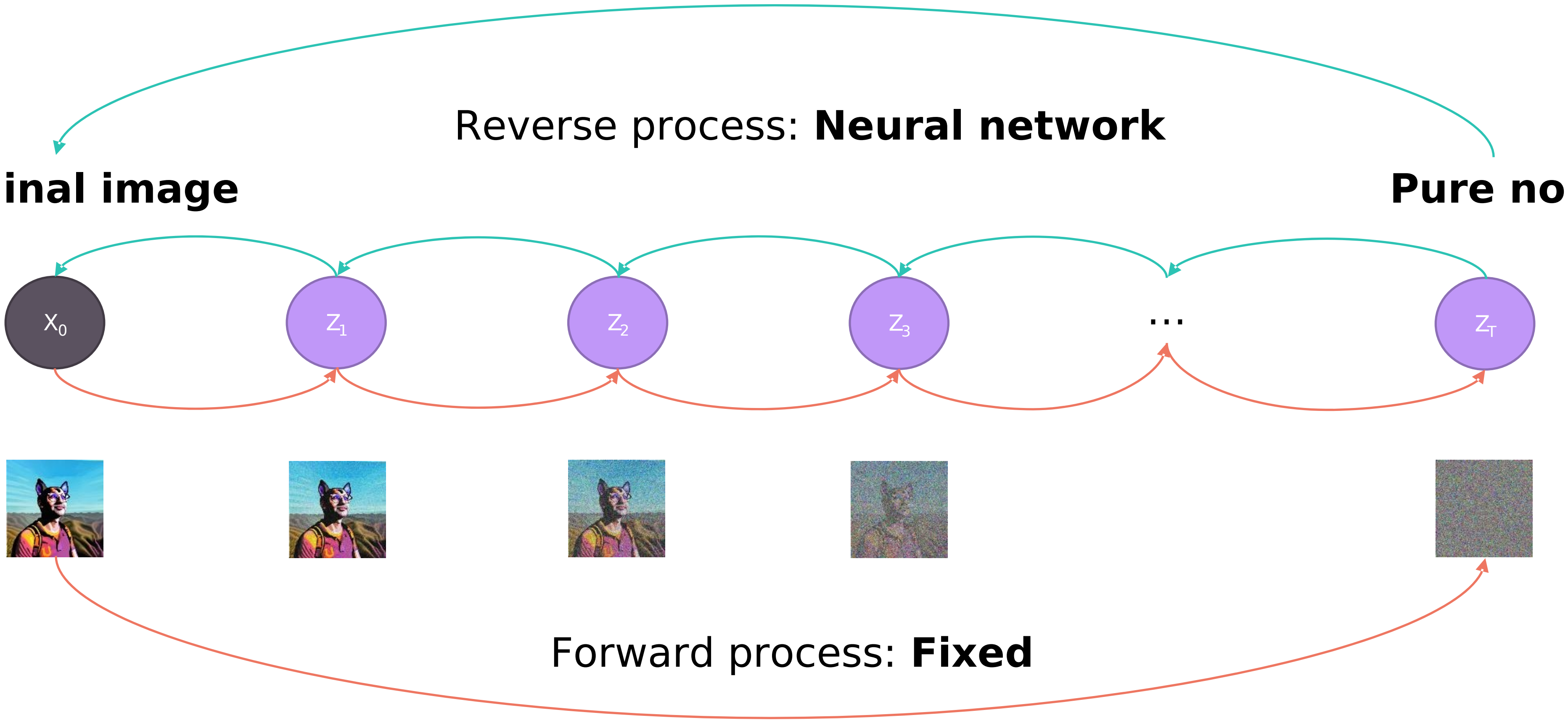
We keep denoising the image progressively for T steps.

U-Net



Ronneberger, O., Fischer, P. and Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18* (pp. 234-241). Springer International Publishing.

How to generate new data?



Reverse process: **Neural network**

Original image

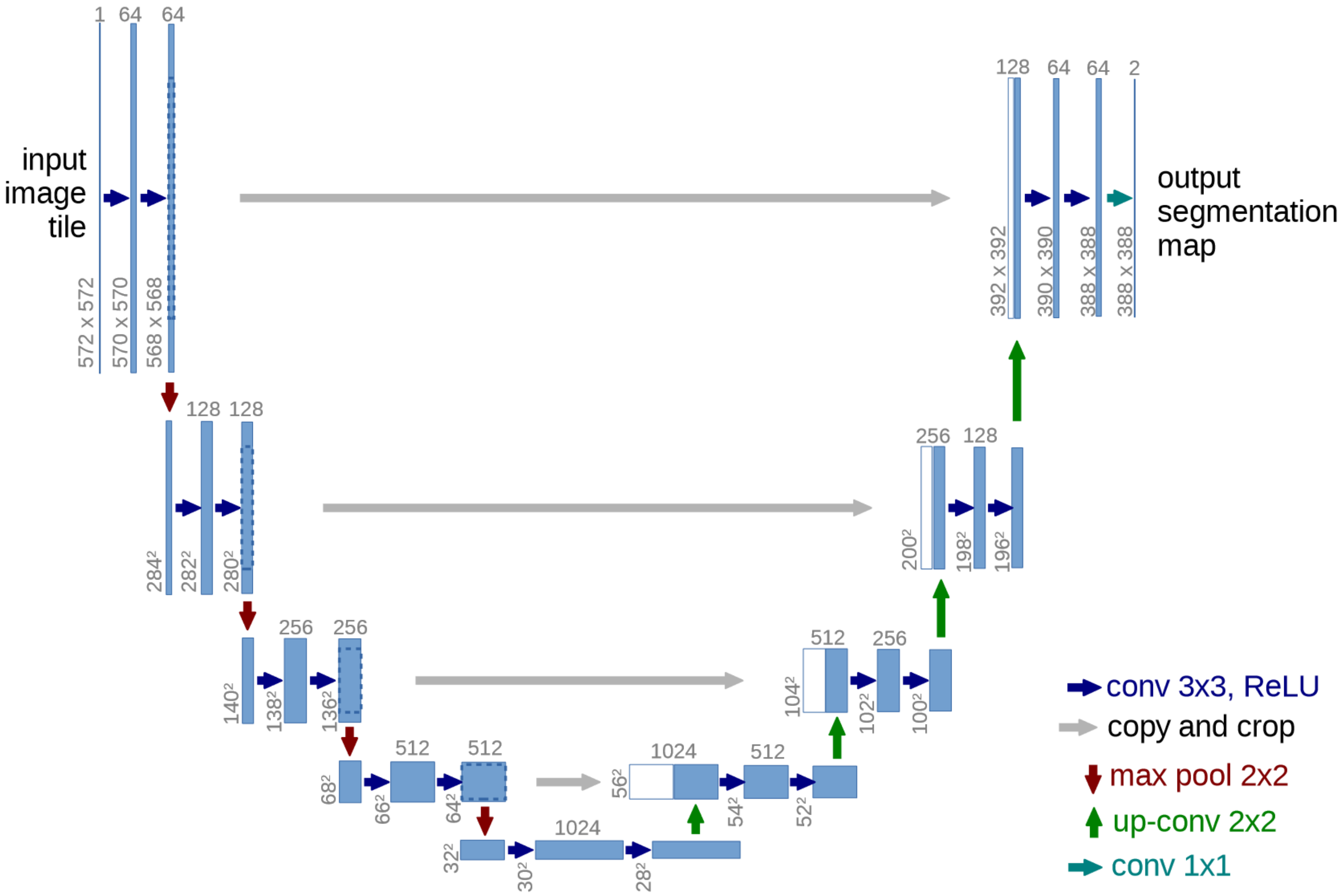
Pure noise

Forward process: **Fixed**

How to condition the reverse process?

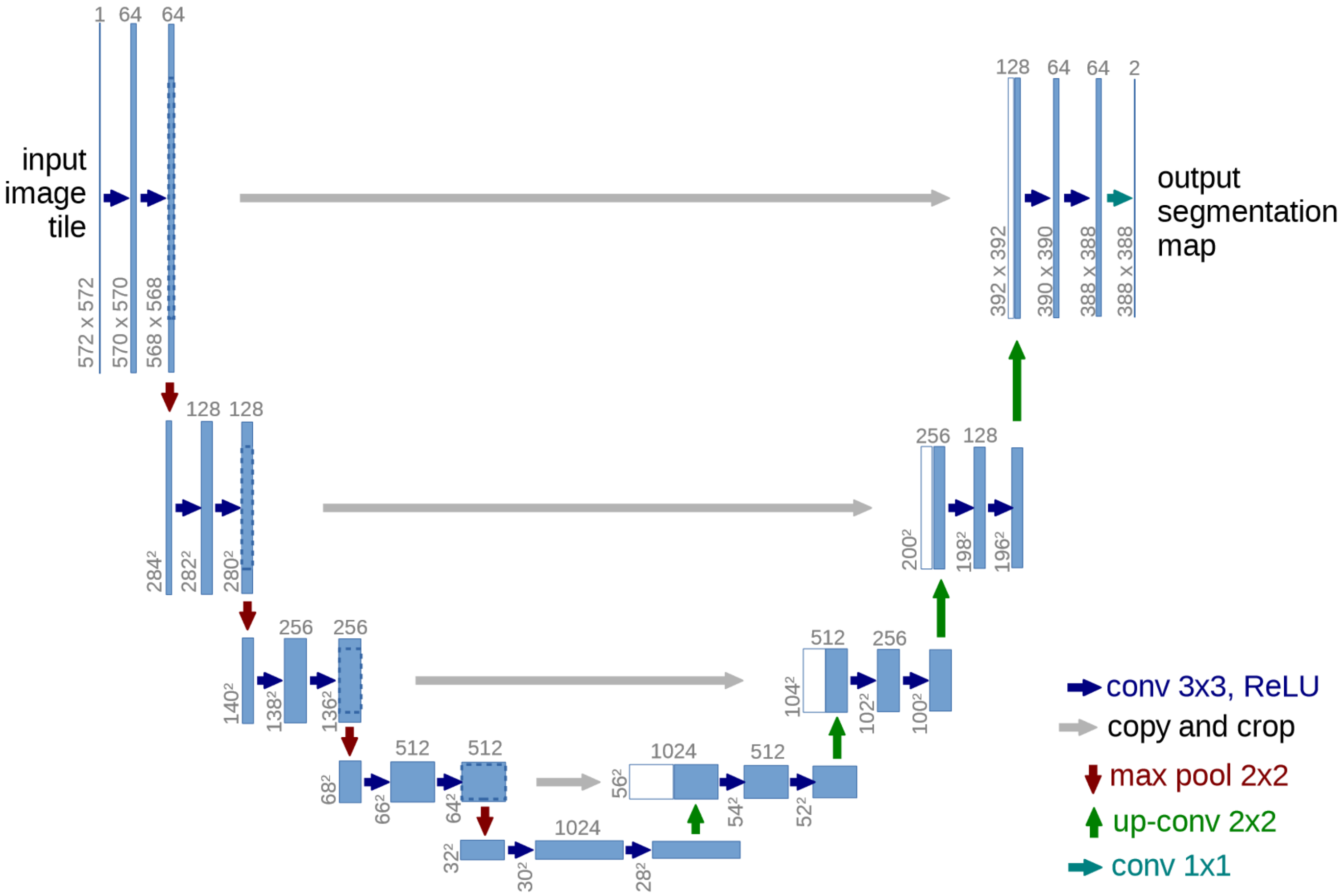
- Since we start from noise in the reserve process, how can the model know what we want as output? How can the model understand our prompt? This is why we need to condition the reverse process.
- If we want to condition our network, we could train a model to learn a joint distribution of the data and the conditioning signal $(,)$, and then sample from this joint distribution. This, however, requires the training of a model for each separate conditioning signal.
- Another approach, called **classifier guidance**, involves the training of a separate model to condition the output.
- The latest and most successful approach is called **classifier-free guidance**, in which, instead of training two networks, one conditional network and an unconditional network, we train a single network and during training, with some probability, we set the conditional signal to zero, this way the network becomes a mix of conditioned and unconditioned network, and we can take the conditioned and unconditioned output and combine them with a weight that indicates how much we want the network to pay attention to the conditioning signal.

Classifier Free Guidance (Training)



Ronneberger, O., Fischer, P. and Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18* (pp. 234-241). Springer International Publishing.

Classifier Free Guidance (Inference)



Ronneberger, O., Fischer, P. and Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18* (pp. 234-241). Springer International Publishing.

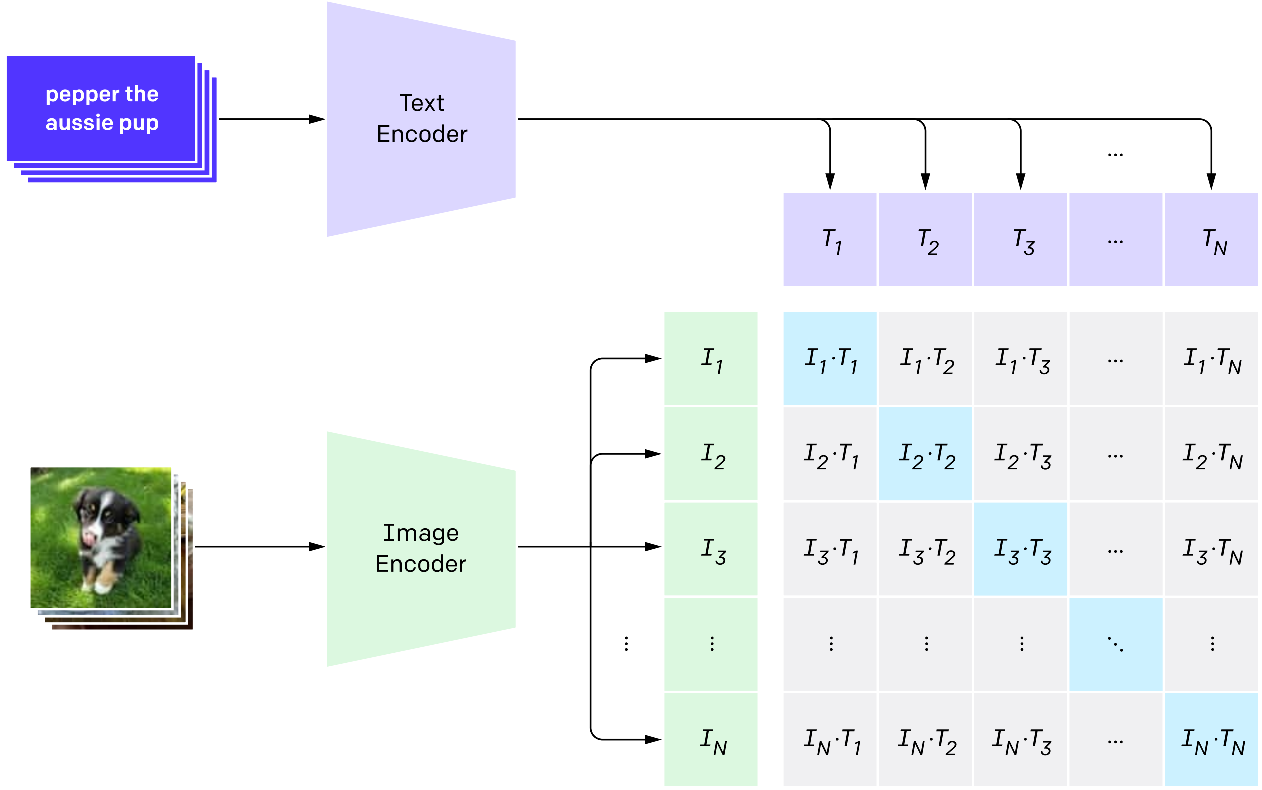
Classifier Free Guidance (Combine outpu

$$o = * \; (\; iti \; - \; iti \;) + \; iti$$

A weight that indicates how much we want the model to pay attention to the conditioning signal (prompt).

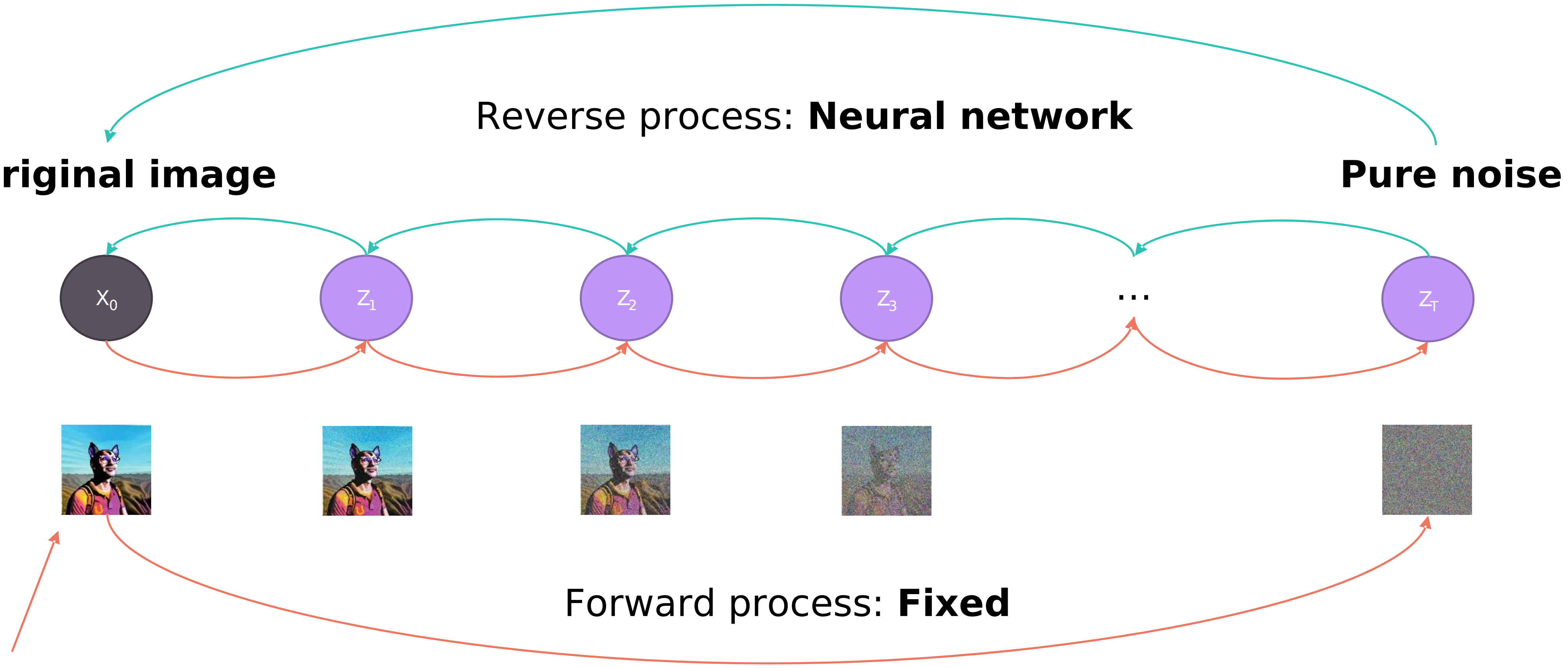
CLIP (Contrastive Language–Image Pre-trainin

1. Contrastive pre-training



	T_1	T_2	T_3	...	T_N
I_1	$I_1 \cdot T_1$	$I_1 \cdot T_2$	$I_1 \cdot T_3$	...	$I_1 \cdot T_N$
I_2	$I_2 \cdot T_1$	$I_2 \cdot T_2$	$I_2 \cdot T_3$	...	$I_2 \cdot T_N$
I_3	$I_3 \cdot T_1$	$I_3 \cdot T_2$	$I_3 \cdot T_3$	...	$I_3 \cdot T_N$
⋮	⋮	⋮	⋮	⋱	⋮
I_N	$I_N \cdot T_1$	$I_N \cdot T_2$	$I_N \cdot T_3$	...	$I_N \cdot T_N$

Performing many steps on big images is



Reverse process: **Neural network**

Original image

Pure noise

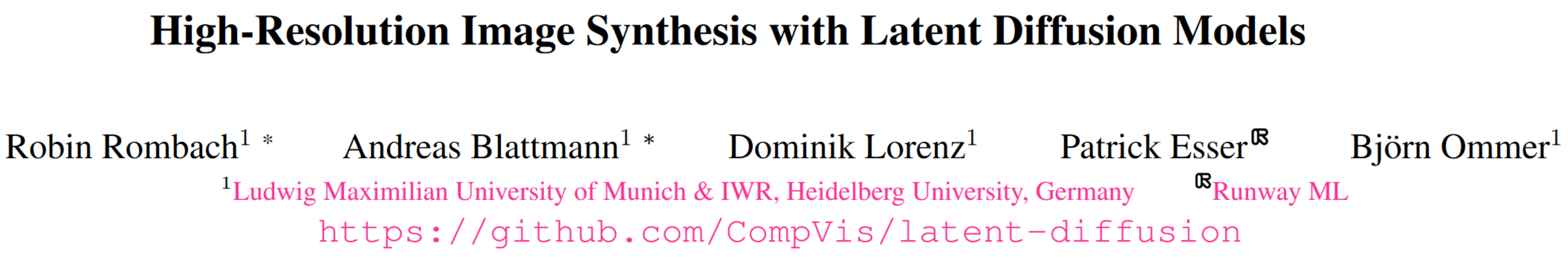
X_0 Z_1 Z_2 Z_3 … Z_T

Forward process: **Fixed**

Since the latent variables have the same dimension (size of the vector) as the original data, if we want to perform many steps to denoise an image, that would result in a lot of steps through the Unet, which can be very slow if the matrix representing our data/latent is large. What if we could **“compress”** our data before running it through the forward/reverse process (UNet)?

Latent Diffusion Model

- Stable Diffusion is a latent diffusion model, in which we don’t learn the distribution p(x) of our data set of images, but rather, the distribution of a latent representation of our data by using a **Variational Autoencoder**.
- This allows us to reduce the computation we need to perform the steps needed to generate a sample, because each data will not be represented by a 512x512 image, but its latent representation, which is 64x64.



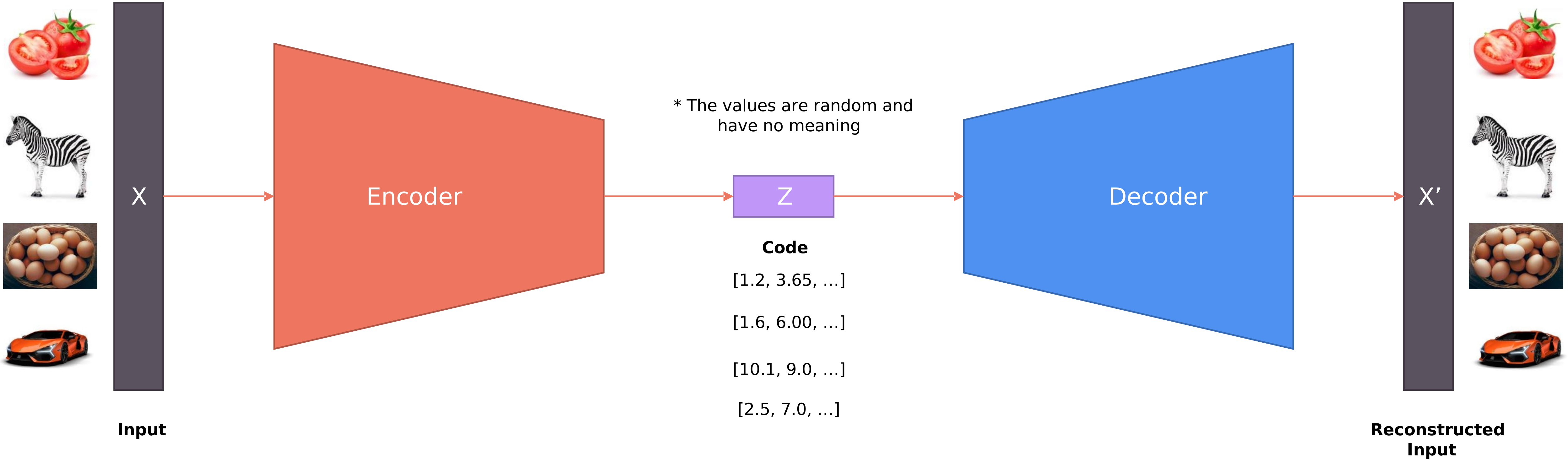
High-Resolution Image Synthesis with Latent Diffusion Models

Robin Rombach[1] * Andreas Blattmann[1] * Dominik Lorenz[1] Patrick Esser Björn Ommer[1]

[1] Ludwig Maximilian University of Munich & IWR, Heidelberg University, Germany Runway ML

https://github.com/CompVis/latent-diffusion

What is an Autoencoder?



X

Encoder

* The values are random and have no meaning

Z

Code

[1.2, 3.65, …]

[1.6, 6.00, …]

[10.1, 9.0, …]

[2.5, 7.0, …]

Decoder

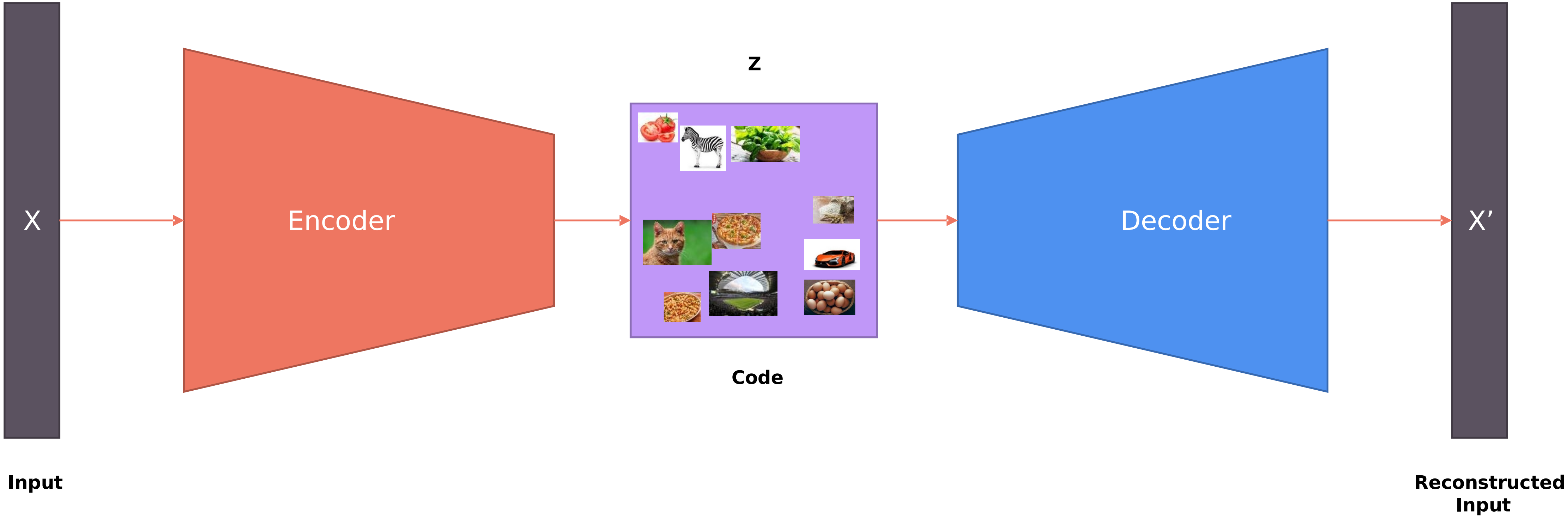
X’

Input

Reconstructed Input

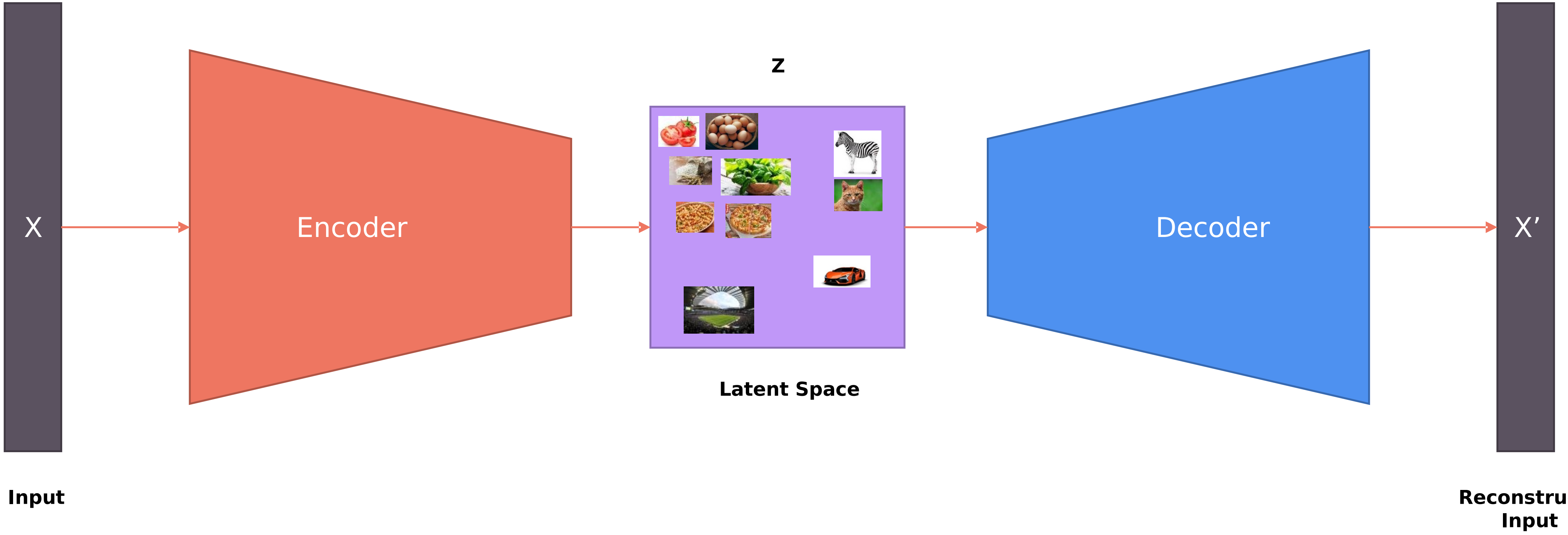
What’s the problem with Autoencoders?

The code learned by the model **makes no sense**. That is, the model can just assign any vector to the inputs without the numbers in the vector representing any pattern. The model doesn’t capture any **semantic relationship** between the data.

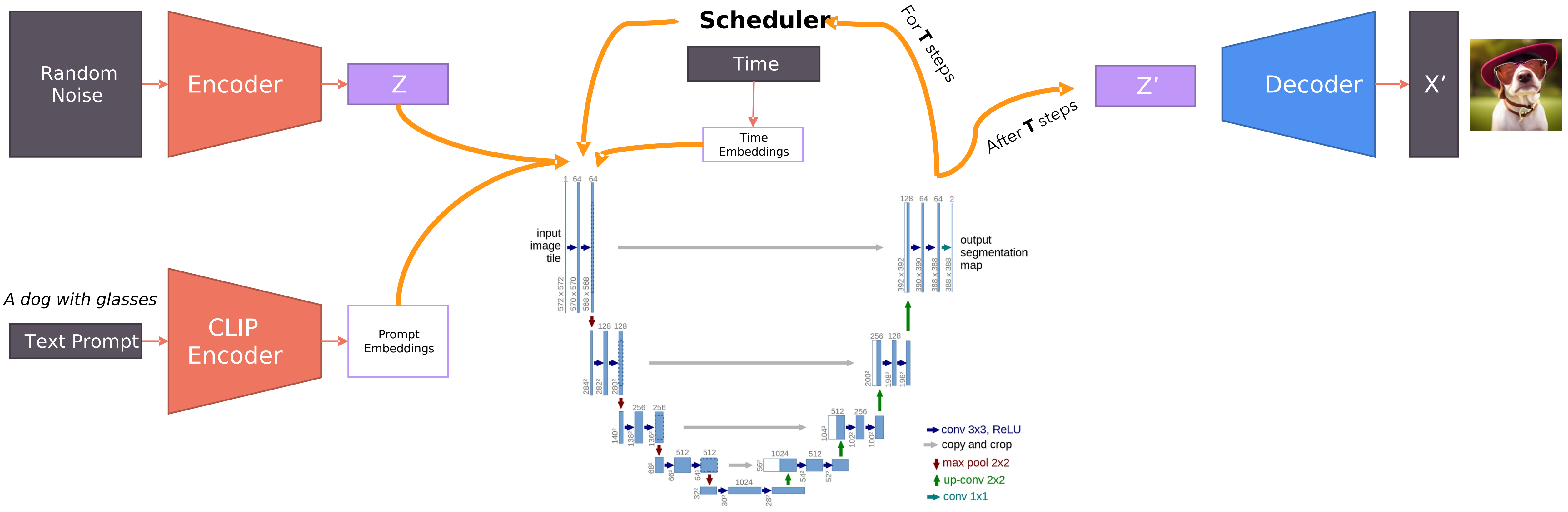


Introducing the Variational Autoencoder

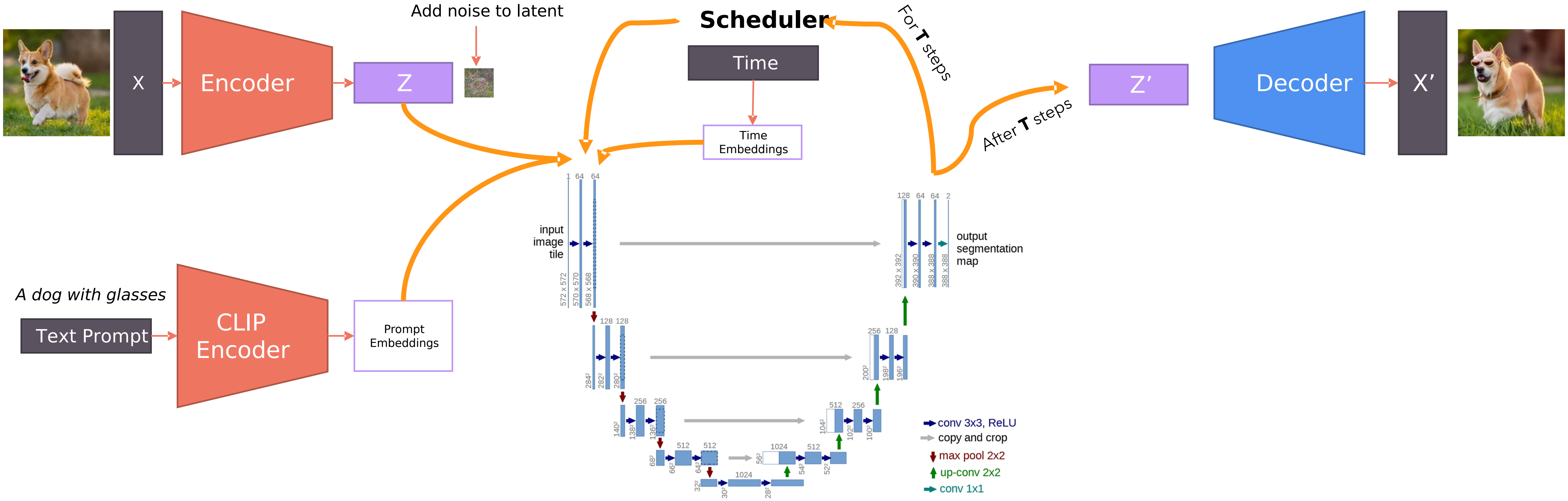
The variational autoencoder, instead of learning a code, learns a “**latent space**”. The latent space represents the parameters of a (multivariate) distribution.



Architecture (Text-To-Image)



Architecture (Image-To-Image)



X

Encoder

Z

Add noise to latent

Scheduler

Time

Time Embeddings

For **T** steps

After **T** steps

Z'

Decoder

X'

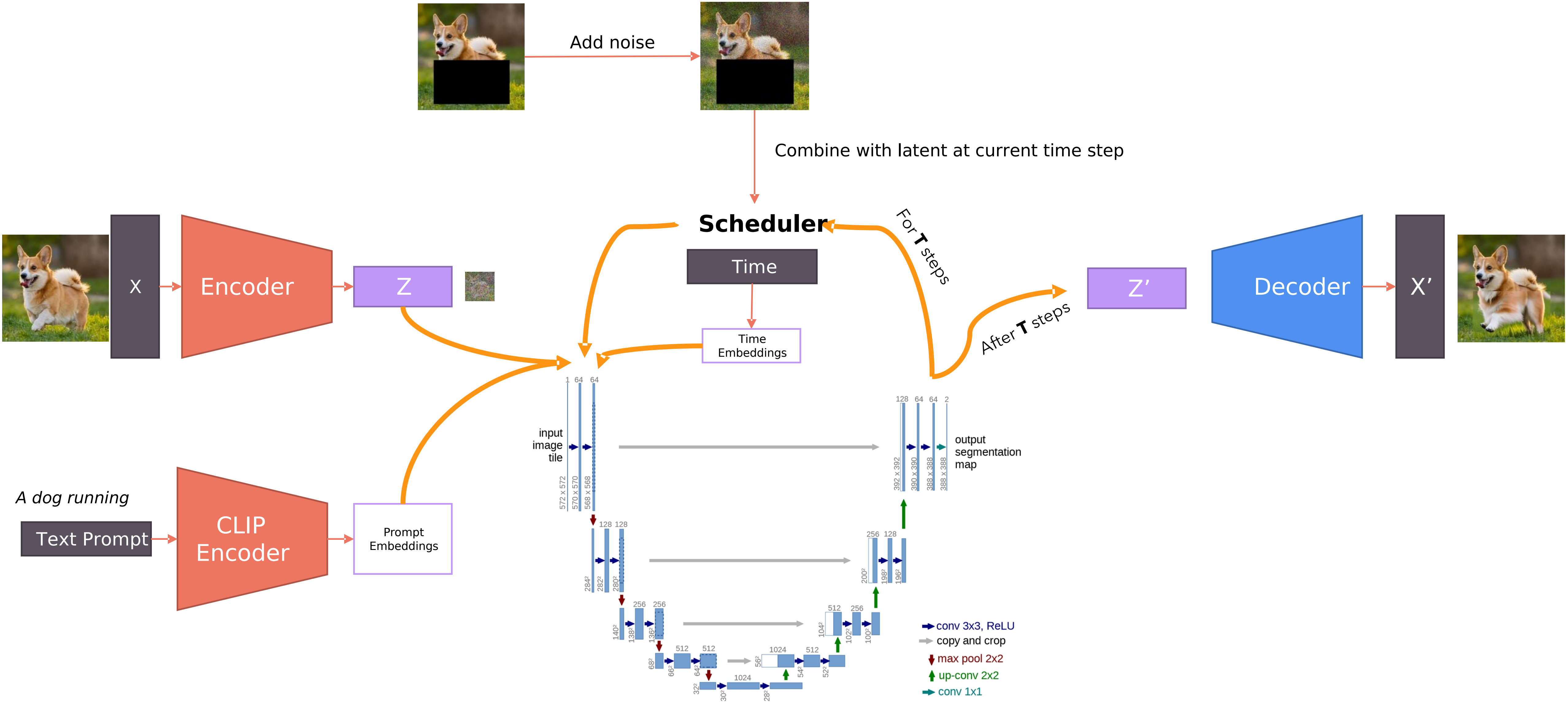
A dog with glasses

Text Prompt

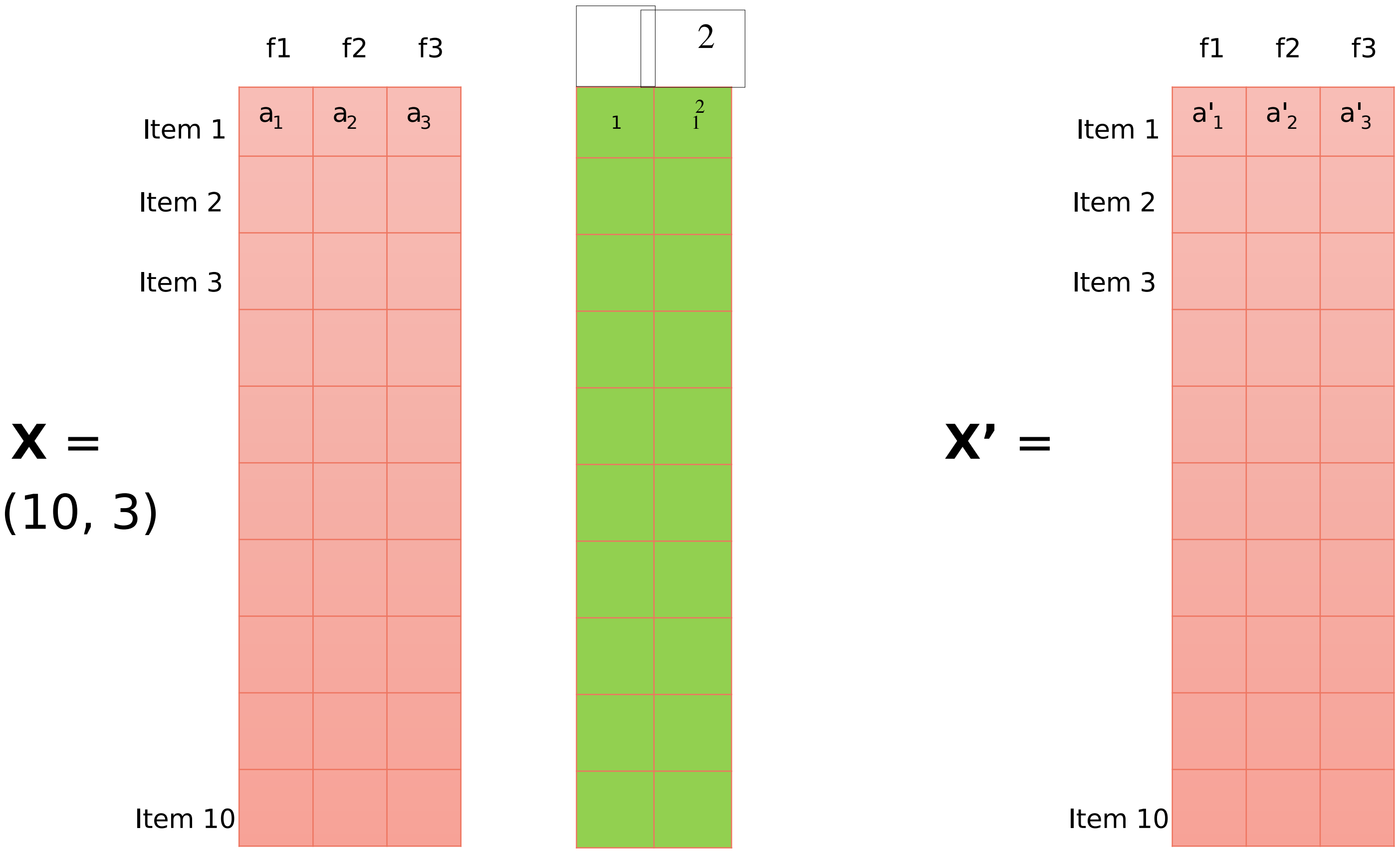
CLIP Encoder

Prompt Embeddings

Architecture (In-Painting): how to fool mo



Layer Normalization



X = (10, 3)

	f1	f2	f3
Item 1	a_1	a_2	a_3
Item 2			
Item 3			
Item 10			

X' =

	f1	f2	f3
Item 1	a'_1	a'_2	a'_3
Item 2			
Item 3			
Item 10			

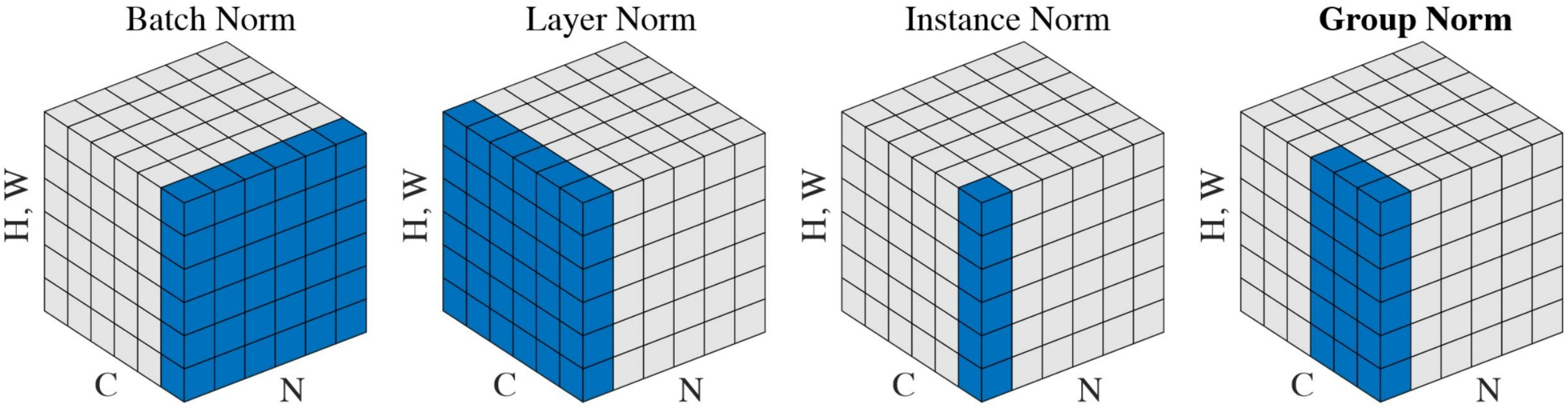
$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

- Each item is updated with its normalized value, which will turn it into a normal distribution with 0 mean and variance of 1.
- The two parameters **gamma** and **beta** are learnable parameters that allow the model to “amplify” the scale of each feature or apply a translation to the feature according to the needs of the loss function.

With batch normalization we normalize by **columns (features)**

With layer normalization we normalize by **rows (data items)**

Group Normalization



Documments

1. https://github.com/CompVis/stable-diffusion/
2. https://github.com/divamgupta/stable-diffusion-tensorflow
3. https://github.com/kjsman/stable-diffusion-pytorch
4. https://github.com/huggingface/diffusers/