

# NumByNum :: High-Resolution Image Synthesis with Latent Diffusion Models (Rombach et al., 2022) Reviewed



Aria Lee · Follow

26 min read · Nov 15, 2023



39



This review of “High-Resolution Image Synthesis with Latent Diffusion Models (Rombach et al., 2022)” begins at Number 1 and concludes at Number 197. I may make additions or revisions to the content in the future for updates. I am creating this post primarily for my personal understanding of the subject, and I humbly acknowledge that errors or inaccuracies may be present. If you happen to identify any such issues, please do not hesitate to bring them to my attention. Thank you, and hope you enjoy 😊!

1. If you think about the most eye-catching field in artificial intelligence applications in recent years besides ChatGPT, you might recall image generation models like DALL-E 2, GLIDE, Imagen, and others.

2. I've been dodging this paper like avoiding a chart-topping hit song 🎵. But alas, before the clock strikes midnight, I found myself compelled to skim through its crucial contents.

3. And why did this sudden urge strike me? Oh, probably because I've got a report due tomorrow...

4. Like clockwork, the most captivating papers unveil themselves precisely when there's a pile of other tasks begging for attention. Typical. 🧑

5. So today, I'm planning to explore image generation models centered around the paper "High-Resolution Image Synthesis with Latent Diffusion Models" (Rombach et al., 2022). This is that **stable diffusion** model I've heard about countless times but resiliently avoided studying.

6. I typically refrain from including 'pre-requisite reading' links since I prefer when I can comprehend all necessary concepts within a single article without the need to refer back and forth with additional links. However, just for this instance as an exception, considering the extensive content involved in understanding the diffusion model, I recommend delving into the previous article that specifically covers DDPM.

7. Of course, I've excerpted the essential parts in the main text, so starting from here wouldn't be a problem. It's the same if you already understand diffusion. Nevertheless, having a grasp of the diffusion concept might make things more comfortable.

8. **Alright, enough digression. Let's get started.**

9. And yes, as always, it's going to be the typical meandering journey. 😊

## 10. Stable diffusion fundamentally pertains to image synthesis.

### High-Resolution Image Synthesis with Latent Diffusion Models

Robin Rombach<sup>1</sup> \*    Andreas Blattmann<sup>1</sup> \*    Dominik Lorenz<sup>1</sup>    Patrick Esser<sup>®</sup>    Björn Ommer<sup>1</sup>

<sup>1</sup>Ludwig Maximilian University of Munich & IWR, Heidelberg University, Germany    <sup>®</sup>Runway ML

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

#### Abstract

*By decomposing the image formation process into a sequential application of denoising autoencoders, diffusion models (DMs) achieve state-of-the-art synthesis results on image data and beyond. Additionally, their formulation allows for a guiding mechanism to control the image generation process without retraining. However, since these models typically operate directly in pixel space, optimization of powerful DMs often consumes hundreds of GPU days and inference is expensive due to sequential evalu-*



## 11. It basically means a model that generates new images.

12. More precisely, it performs tasks such as image generation, class-conditional image synthesis, super-resolution, image inpainting, among others. These tasks are all grouped under models that generate new images using empty or low-quality images.

13. Of course, even before stable diffusion, there were many good image synthesis models. Notably, these can be broadly categorized into 1) GAN-based methods and 2) likelihood-based methods.

14. GAN, as briefly mentioned in the Gaussian Dreamer article or the DDPM article, is a model proposed in the “Generative Adversarial Networks” (Goodfellow et al., 2014). paper.

15. Essentially, it involves a structure where a generator creates fake samples, and a discriminator distinguishes whether these are real or fake, creating more realistic fake images progressively, akin to a forger making better counterfeit notes while the police determine their legitimacy, leading to the forger's improvement.

16. The advantage of GANs is the remarkably high quality of the generated images. Even the early versions of GANs had decent performance, but subsequent research led to the creation of incredibly high-quality samples.

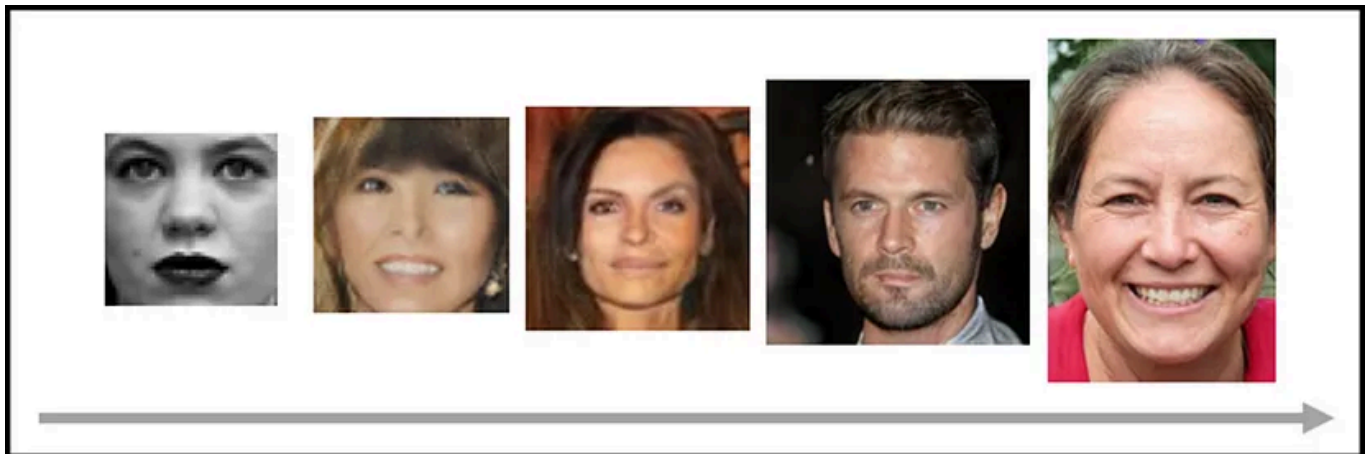


image from [here](#)

17. Despite its good ideas and performance, GANs face two fundamental challenges: 1) they are difficult to optimize, and 2) they struggle to encompass the full data distribution. The detailed explanation for why optimization is challenging is extensively covered in [this](#) article, and we'll address this separately in the future.

18. Additionally, GANs' inability to encompass the entire data distribution means a lack of diversity in generated images. As GANs aim to deceive the discriminator with realistic images, they tend to focus on perfecting just a few images rather than encompassing the entire training set.

19. Essentially, even if the discriminator cannot distinguish all animals well and only accurately depicts cats, there's no issue in deceiving the discriminator.

20. In other words, once the discriminator is deceived successfully, the model tends to focus solely on generating images that have already been classified as real, leading to what's known as the “mode collapse” issue.

21. This is what we call the “mode collapse” problem in GANs.

22. Alright, I get GANs now. So, what about the second category, the likelihood-based method?

23. To understand this, let's briefly review Ian Goodfellow's famous taxonomy of generative models.

## Taxonomy of generative models

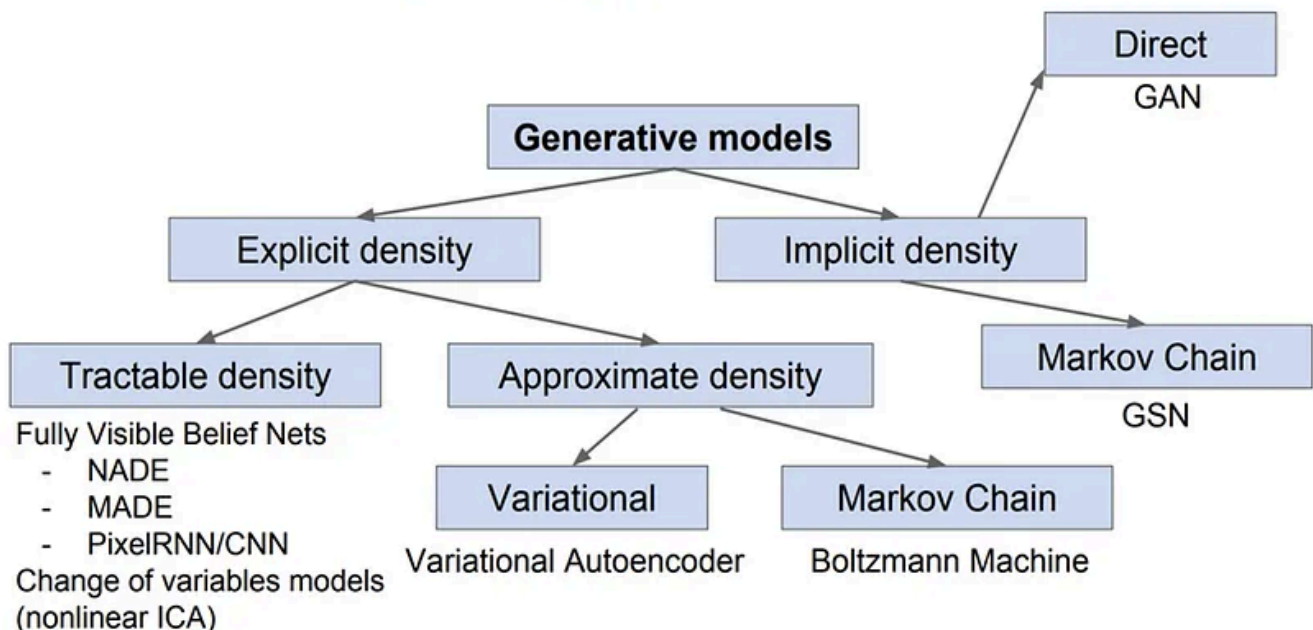


image from [here](#)

24. There might be some confusing terms, but when dissected, they aren't that complicated. **Essentially, the top of the generative models divides into explicit density and implicit density.**

25. Generating natural images fundamentally involves deriving a distribution of existing natural images and then picking one sample from this distribution.

26. For instance, if we want to generate cat photos, we somehow acquire the highly-dimensional distribution of actual cat photos in existence. Then, pulling one sample from this distribution would result in a natural-looking photo.

27. Depending on how we acquire this extremely high-dimensional natural distribution, we divide into explicit density and implicit density. Explicit density involves directly determining the distribution using any method, while implicit density involves approximating the distribution by appropriately sampling samples.

28. Hence, GANs belong to implicit density. They don't explicitly define the probability distribution but operate by sampling from the distribution created by the generator.

29. On the other hand, models explicitly defined belong to tractable density and approximate density. Tractable density assumes a computable probability distribution for the high-dimensional natural distribution we want to acquire.

30. In contrast, approximate density aims to approximate the natural distribution using a more complex, intractable distribution than the

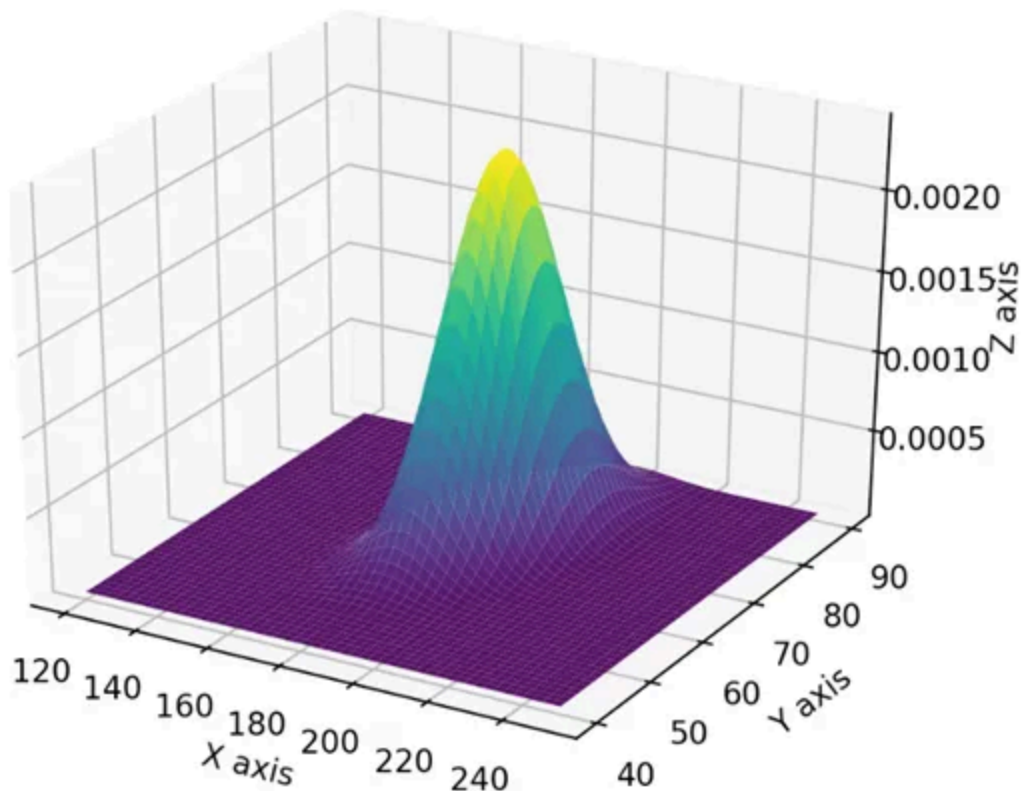


computable one.

31. Now, back to likelihood-based models. Since our paper separates generative models into GANs and likelihood-based ones, based on Ian Goodfellow's taxonomy, excluding GANs, it seems to categorize likelihood-based methods into autoregressive, flow-based, and prescribed models.

32. These models try to approximate the extremely high-dimensional distribution of 'natural images' and generate samples. But what does 'generate samples' mean?

33. Generating (the most natural) samples means, ultimately, sampling from the probability distribution we have acquired, specifically from the point where the likelihood (probability) of naturalness is highest.



34. For example, if the probability distribution looks like the figure above, it would be more natural (i.e., with a higher likelihood) to sample from the central area than from the four corners.

35. By pinpointing positions on the extremely high-dimensional probability distribution corresponding to images, and selecting the positions with the highest probability (likelihood), it's generating samples from the most 'likely' point. **Thus, models falling into this category are called likelihood-based models.**

36. Likelihood-based models can roughly be categorized again into about three representative branches.

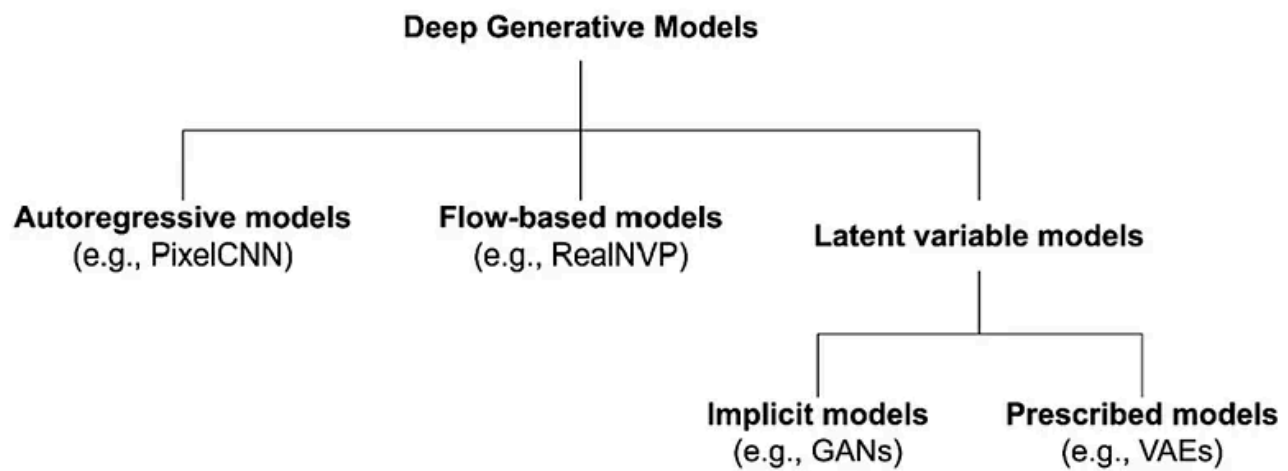


image from [here](#)

37. The diagram categorizing generative models slightly differs from Ian's; simply put, **explicit density models branch into 1) autoregressive models, 2) flow-based models, and 3) prescribed models.** In our paper's terms, likelihood-based models are further classified into autoregressive, flow-based, and prescribed models.



38. Let's take a quick look at each. You can refer to [this](#) article for further information.

39. **First off, in the realm of likelihood-based models, we have autoregressive models.**

40. When in a temporal sequence, the current state relies on past states to represent the current value as a function of past values. This probability process is known as an autoregressive process. It's a method to predict future values based on past values.

41. Autoregressive models express the distribution we aim to obtain using the following equation.

$$p(\mathbf{x}) = p(x_0) \prod_{i=1}^D p(x_i | \mathbf{x}_{<i}),$$

42. In essence, it means computing the current step's value by multiplying all the previous steps. While density estimation may work well, calculating each step sequentially requires significant computation. Moreover, since the future relies on the past, sampling must proceed sequentially and follow a specific order.

43. **The second branch of likelihood-based models consists of flow-based models.**

44. These models take an input image  $x$  into a flow function  $f$  to obtain latent  $z = f(x)$ . Then, using the inverse flow function, they pass  $z$  to reconstruct the

original image.

45. By continually applying the inverse transformation function to a very simple distribution, flow-based models reach the initial probability distribution we desire. It's another important concept, which we will cover separately in another article. If you're curious now, refer to [this](#) well-explained article.

46. **The third category is prescribed models, among which VAE (Variational Autoencoder) holds significance.** There's an informative [article](#) on this topic that I'll summarize here.

47. Imagine entering a room and finding a strand of hair with a length  $x$ . In this scenario, we need to predict whether this hair belongs to a male or a female. We have two choices.

48. The first option compares the probability of the length  $x$  given it's from a male,  $p(x|\text{male})$ , and given it's from a female,  $p(x|\text{female})$ . Then, it chooses the gender with the higher probability.

49. The second option compares the probability of the owner being a male given the length  $x$ ,  $p(\text{male}|x)$ , and the probability of the owner being a female given the length  $x$ ,  $p(\text{female}|x)$ . Then, it selects the gender with the higher probability.

50. They might seem similar, but there's a significant difference.

51. If we assume the population in that area is 100% male, the second option acknowledges the sex ratio, knowing that  $p(x|\text{female})$  is 0. However, the first

option still solely calculates the probability of having hair length  $x$  for males and females, disregarding the sex ratio.

52. So, even though the second option, which considers the sex ratio, would provide more accurate results, we're in a situation where we have no information about the sex ratio and hence have to opt for the first method.

53. Likelihood-based generative models, needing to approximate a similar distribution without knowing the answer to the immensely high-dimensional target distribution, face a similar situation.

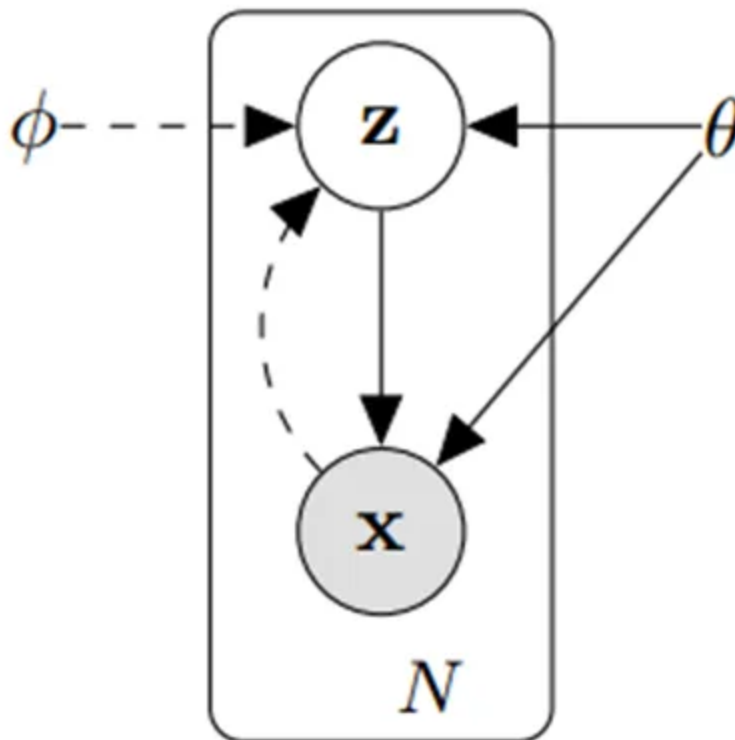


image from [here](#)

54. Let's look at the diagram. The  $x$  within the gray circle represents the natural images we have, and  $z$  is the latent variable (which we don't know but

generates image  $x$ ). If there are a total of  $N$  images, there would be  $N$  pairs of latent variables  $z$  and images  $x$ .

55. Our aim is to observe natural image  $x$  and determine the distribution of  $z$  (a highly dimensional natural distribution) that generated this image.

56. However, there's a problem. What we want is  $p_{\theta}(z|x)$ , and to compute this, we need to calculate  $p_{\theta}(x|z) * p_{\theta}(z) / p_{\theta}(x)$ .

57. But if we already knew the latent distribution  $p(z)$  or the image's distribution  $p(x)$ , there would be no need for such computations. We would simply sample from that known distribution since we know the correct distribution.

58. This situation is precisely what was mentioned as 'intractable' in Num 30.

59. If you've followed up to this point, you've understood the problem VAE aims to solve. VAE stands for Variational AutoEncoder, intending to tackle a simpler problem (variational) because directly calculating the posterior is too challenging.

60. Variational methods usually employ the following lower bounding technique.

$$\mathcal{L}(x; \theta) \leq \log p_{\text{model}}(x; \theta).$$

61. The concept is straightforward. Since we can't directly obtain the real distribution  $p(x; \theta)$  on the right-hand side, we set a lower bound with

another model  $L(x; \theta)$  that we know and can compute, then maximize this lower bound to maximize the right-hand side.

62. Thus, VAE selects a model  $Q_\phi$ , which we know well and can compute. It adjusts the parameters  $\phi$  to make  $Q_\phi$  as close as possible to the desired posterior distribution  $p$ .

63. The KL divergence is used to bring these two distributions closer together.

64. In simple terms, KL divergence is a value that represents the difference between two probability distributions.

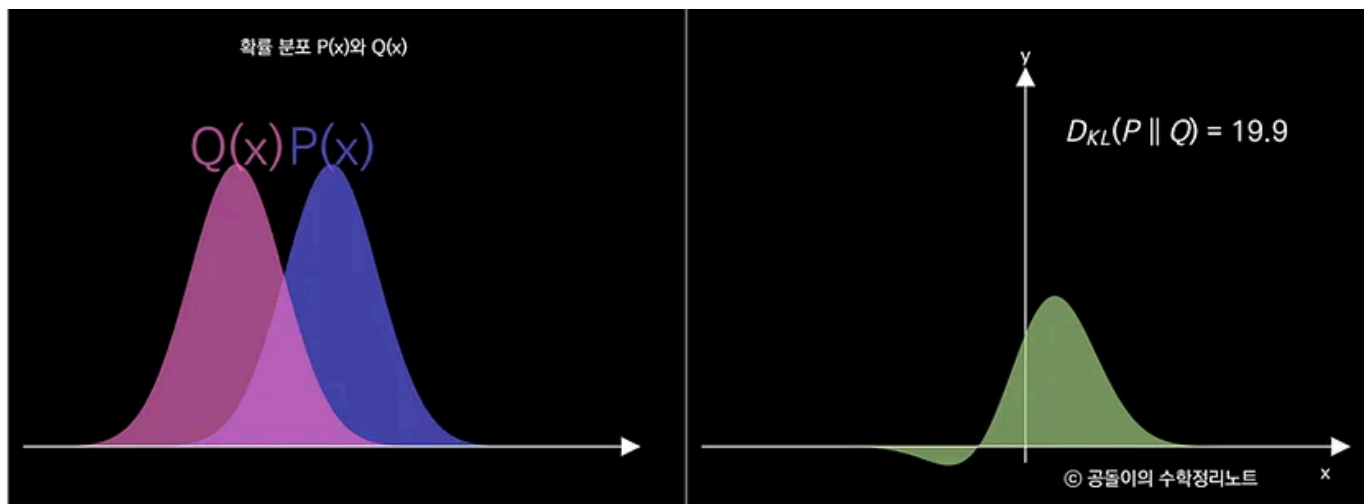


image from [here](#)

65. If we have probability distributions  $Q(x)$  and  $P(x)$  as shown in the picture, the difference between these two distributions, known as KL divergence, can be visualized as the area of the green function on the right. For continuous distributions, KL divergence can be expressed in the formula below.

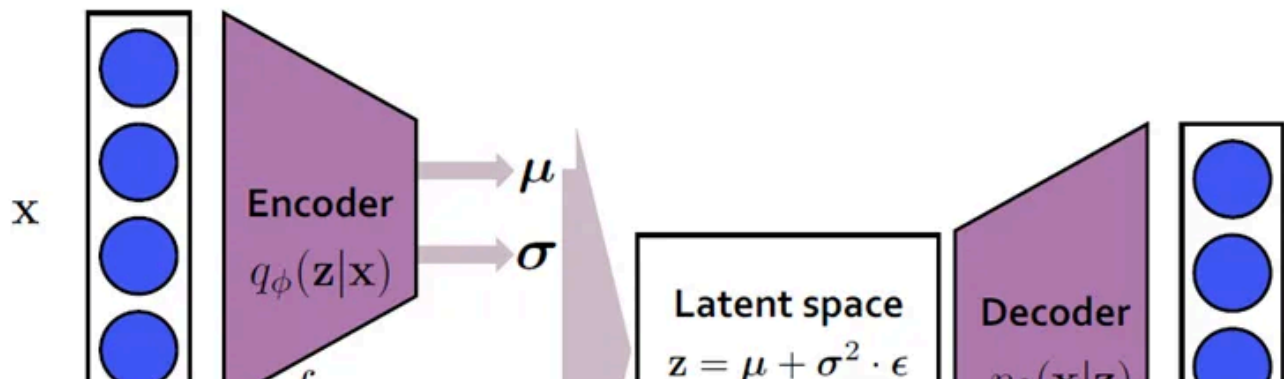
$$D_{\text{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \left( \frac{P(x)}{Q(x)} \right)$$

66. I'll skip over why the equation looks like this since it's covered in the [DDPM article](#).

67. Anyway, to approximate the posterior  $p$  like this, adjusting the parameters  $\phi$  of a simpler distribution  $q$  while reducing the KL divergence between  $p$  and  $q$  is known as variational inference. That's why VAE becomes a “variational” autoencoder.

68. In other words, variational inference is a powerful tool because it transforms the stochastic process of estimating posterior into an optimization problem of optimizing  $\phi$ , thereby reducing the KL divergence between  $p$  and  $q$ .

69. To optimize this  $\phi$ , VAE utilizes the encoder and decoder structures, hence becoming a variational “autoencoder”.



Open in app ↗

Sign up

Sign in

Medium

Search

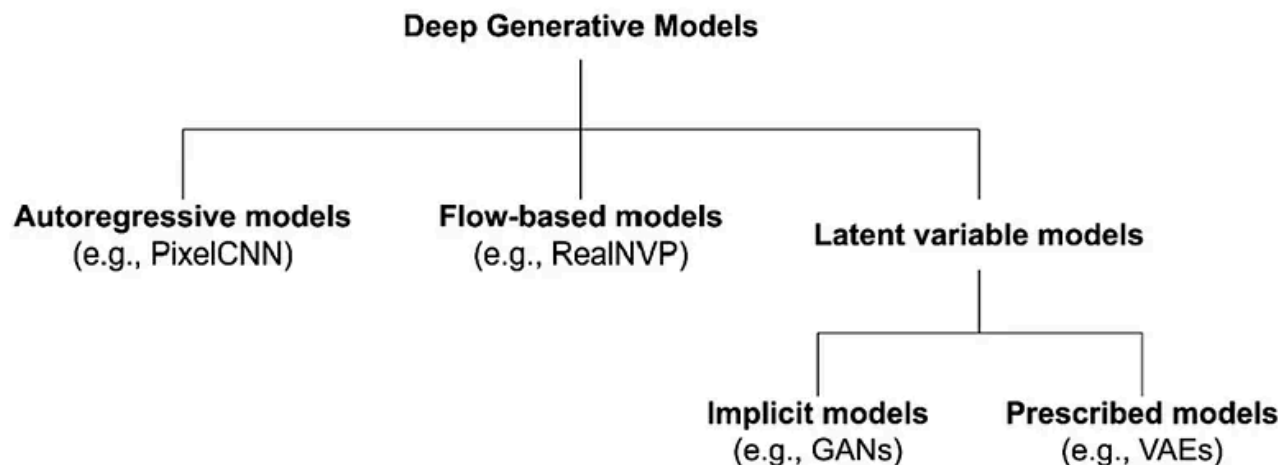
Write



image from [here](#)

70. Covering this structure might make it easier to understand, but it's becoming lengthy. Since VAE will be addressed separately in another article in the future, I'll stop here for now. Anyway, summarizing, VAE converts the posterior estimation problem using variational inference techniques into a  $\phi$  optimization problem, which it proceeds to solve using an autoencoder structure.

71. So far, we've divided likelihood-based models into autoregressive, flow-based, and variational autoencoder categories.



72. Comparing briefly, GAN and likelihood-based models in the above diagram yield the following differences.



image from [here](#)

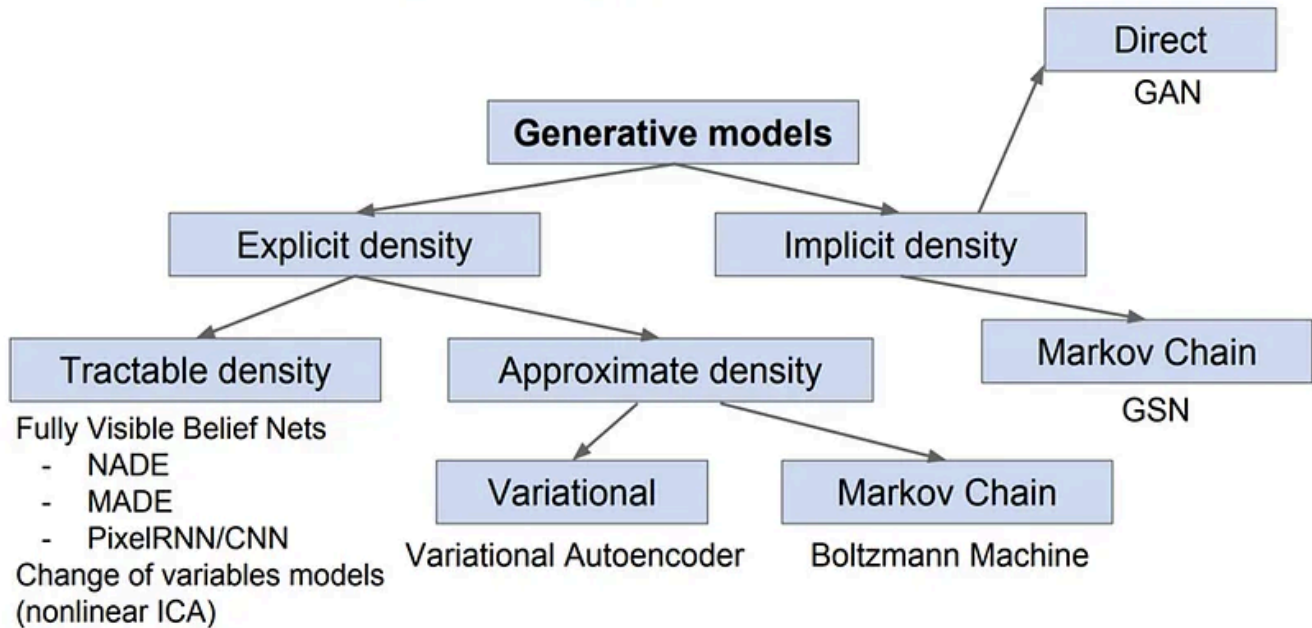
73. The drawbacks of VAE and flow-based models both lie in the slightly reduced quality of generated images compared to GAN. **Additionally, likelihood-based models commonly require directly calculating maximum likelihood, resulting in enormous time and computation costs for training.**

74. While they don't suffer from mode collapse like GAN, they face the opposite problem of requiring excessive resources to estimate highly complex distributions.

75. "Because pixel based representations of images contain barely perceptible, high-frequency detail, **maximum-likelihood training spends a disproportionate amount of capacity on modeling them, resulting in long training times.**"

76. If you've followed along this far, you've roughly covered the concepts of representative generative models.

# Taxonomy of generative models



77. Among the models, there's GAN for indirect estimation of the distribution to be modeled via sampling, while autoregressive models, flow-based models, and VAE lie within the explicit density category for directly estimating the distribution.

78. Here's one more thing.

79. The diffusion model we explored in the [DDPM article](#) also belongs to this likelihood-based method.

80. For those who haven't read the previous article, let's briefly review essential parts about diffusion. If you've already read it or are familiar with diffusion, feel free to skip ahead to Num105! 😊🚀

81. In 2015, a paper titled "[Deep Unsupervised Learning using Nonequilibrium Thermodynamics](#)" (Sohl-Dickstein et al., 2015) was published.

82. You might wonder why a deep learning paper suddenly mentions the word “thermodynamics.”

83. That’s because it drew inspiration from Brownian motion.

84. Let’s imagine dropping a clump of pollen into a pond, for instance. Initially, shortly after dropping it, the particles will cluster together, but over time, they will gradually disperse and spread out. When particles scatter in a liquid or gas, following random motion, it’s referred to as Brownian motion.

85. Looking at the illustration above, you can see that the particles inside the yellow circle don’t move in a predetermined direction but take random steps, guided by specific probability distributions each time. This gradual, random spreading of particles is called diffusion.

86. Now, if you wanted to revert the fully dispersed, nearly random state of pollen back to its initial round shape, how would you do it?

87. You’d have to retrace the steps of random motion taken so far in reverse, effectively going against the diffusion process. By continually going in reverse, you could eventually return to the initial state.

88. This idea, applied directly to image generation, is the basis of the diffusion model.

89. First, prepare a pristine image. This image represents the initial state of the pollen's dispersion.

90. Now, add Gaussian noise to the image over 1,000 steps. After 10 steps, you can still recognize the original image, but after around 500 steps, it becomes increasingly challenging to identify the details. Adding noise continuously for 1,000 steps will render the original image completely unrecognizable.

image from [here](#)

91. From the leftmost image in the illustration above, we progressively add Gaussian noise at each step, eventually transforming it into a completely noisy Gaussian image.

92. This is the forward diffusion process, where the original image is diffused into a Gaussian image.

93. The process mirrors the gradual dispersion of pollen through random motion.

94. Now, what we want to do is straightforward.

95. Since we've defined the forward diffusion process from the cat to noise, we'll create a reverse process to go from noise to the cat. We'll insert any noise and extract the cat image.

96. The paper's idea was to learn a method to remove noise layer by layer, allowing for the generation of a natural original image regardless of the noise.

97. Therefore, papers in the diffusion family typically involve 1) defining the forward process from the original image to a complete noise, and 2) defining the posterior  $q(x_{t-1} | x_t, x_0)$  for removing one step of noise. Then, 3) training the backward model  $p_\theta$  to approximate the true posterior and 4) setting the loss function to make them close.

98. DDPM defines these four components as follows:

99. Similar to the 2015 version of diffusion, DDPM defines the forward process through a Markov chain. It also uses diffusion rate  $\beta$  to add noise, but the difference is that it introduces a shortcut to create  $x_t$  directly from  $x_0$  using  $\alpha$ .

100. Then, it defines a posterior that performs one-step denoising given  $x_t$  and  $x_0$ .

101. The backward process is defined, and instead of directly predicting  $\mu$  and  $\sigma$ , we express them as a function of  $\epsilon$ , which is then estimated by a neural network.

image from [here](#)

102. Consequently, the loss function has been simplified for epsilon.

103. In summary, the algorithm can be described as follows.

104. So, summarizing the entire flow as follows:



image from [here](#)

105. We now reviewed only the essential parts of the diffusion model covered in the previous article.

106. However, the diffusion model also suffers from the same drawbacks as other likelihood-based methods, requiring long training times and heavy computational costs. Let's look at the details:

107. **“DMs belong to the class of likelihood-based models, whose mode-covering behavior makes them prone to spend excessive amounts of capacity (and thus compute resources) on modeling imperceptible details of the data. Although the reweighted variational objective aims to address this by undersampling the initial denoising steps, DMs are still computationally demanding, since training and evaluating requires repeated function evaluations (and gradient computations) in the high-dimensional space of RGB images. As an example, training the most powerful DMs often takes hundreds of GPU days (e.g. 150–1000 V100 days)**

and repeated evaluations on a noisy version of the input space render also inference expensive, so that producing 50k samples takes approximately 5 days on a single A100 GPU.”

108. As both GAN-based and likelihood-based methods have their weaknesses, research naturally emerged to combine several individual models to complement each other.

109. Among these, prominent models include the famous VQ-VAE and VQGAN, both attempting to reduce computational loads in generative models.

110. For instance, the VQ-VAE2 mentioned in our paper is thoroughly described in the paper “Generating Diverse High-Fidelity Images with VQ-VAE-2” (Razavi et al., 2019).

111. Let’s briefly touch on what’s necessary here.

112. Earlier, we mentioned that VAE sends data to the latent space using an autoencoder.

113. However, is the latent space required to be continuous?

114. The idea behind VQ-VAE suggests it doesn't have to be necessarily continuous.

115. “In this paper we use ideas from lossy compression to relieve the generative model from modeling negligible information. **Indeed, techniques such as JPEG have shown that it is often possible to remove more than 80% of the data without noticeably changing the perceived image quality. We compress images into a discrete latent space by vector-quantizing intermediate representations of an autoencoder.**”

116. Therefore, VQ-VAE aims to change the latent space of VAE into a discrete one.

117. The method is simple: create a codebook consisting of several vectors, then when a vector comes out of the encoder, calculate the distance between this vector and all the vectors in the codebook. Now, find the closest (similar) vector in the codebook to the encoder's output vector and feed it into the decoder, transforming the continuous latent into a discrete one.

118. For example, when the encoder converts an image into a  $32 \times 32 \times 1$  vector and passes it to a codebook consisting of 512 vectors, each grid will be transformed into a specific vector from the codebook. As a result, you could obtain  $512^{(32 \times 32)}$  different latents. Now, send these latents to the decoder to generate  $512^{(32 \times 32)}$  different images.

119. If you want a more detailed understanding, you can refer to the paper that proposed VQ-VAE, “Neural Discrete Representation Learning” (Oord et al., 2017). Anyway, by discretizing the latent space in this way, we can efficiently encode and decode large images while maintaining sample quality.

120. VQGAN starts with a similar goal. Proposed in the paper “Taming Transformers for High-Resolution Image Synthesis” (Esser et al., 2020), VQGAN combines the basic structure of VQ-VAE with the strengths of CNNs and transformers.

121. Let's briefly consider CNNs and transformers. CNNs are strong in locality since they view a few neighboring pixels together. On the other hand, transformers can perceive the global information by viewing the entire image.

122. Taking inspiration from this aspect, VQGAN operates by using CNNs in the first stage to learn a codebook that reflects locality well. Then, in the second stage, it uses transformers that capture global information to generate images. Let's keep things simple for each.

123. First is the stage of learning the codebook using CNNs.

124. Similar to VQ-VAE, after comparing the vectors output by the encoder with the codebook vectors, we create a quantized vector  $z_q$  using the closest vectors from the codebook. In the figure, the white box labeled as codebook  $Z$  represents the codebook, and the resultant  $z_q$  is the colorful cube in the bottom right.

125. Now, input this quantized latent vector  $z_q$  into the decoder to reconstruct the original image. Then, determine if the created sample, at the patch level, is real or fake. This is used to calculate the loss, which is backpropagated to learn the suitable codebook for the image.

126. Now, the second stage uses transformers to perform image synthesis.

127. In the figure, you can refer to the transformer part in the gray box.

128. The Transformer predicts the next patch vector autoregressively when given the preceding patch vectors. Yes, this is equivalent to the autoregressive process mentioned in Num 40. It predicts the current patch by considering the previous patches, ultimately generating the image.

129. We'll leave the detailed information for later and stop here for now.

130. **Let's return to the discussion on stable diffusion.**

131. **Up to now, we've divided existing image synthesis models into several branches.** Notably, GAN-based methods and likelihood-based methods, which further split into variational autoencoders, flow-based models, autoregressive models, and diffusion models. We've also examined models like VQ-VAE and VQGAN to address each one's shortcomings.

132. Now, let's clear our minds for a moment and focus solely on likelihood-based methods.

133. In Num 73, we mentioned that likelihood-based models require significant time and computational cost for training due to their need to directly calculate maximum likelihood. Considering these models generate images at the pixel level, training extensively to reach the maximum-likelihood point in a high-dimensional RGB image space is a necessity.

134. In other words, the training time and computational cost are exceptionally high because likelihood-based methods perform synthesis at the ‘pixel level’ and deal with the high-dimensional space of RGB images.

135. This issue primarily arises because existing models directly perform optimization in the pixel space. Handling likelihood calculations in the extremely high-dimensional space becomes burdensome.

136. Our paper aimed to address this very problem. **Instead of performing diffusion in high-dimensional pixel space, the authors proposed conducting diffusion in a reduced-dimension latent space.**

137. To do that, we first need to understand how the pixel space of the diffusion model trained using conventional methods looks. This way, we can find a suitable replacement in a new space.

138. Let’s take a look at the diagram below:



139. This figure represents the rate-distortion trade-off when compressing an already trained diffusion model. You might wonder why the sudden appearance of rate on the horizontal axis and RMSE on the vertical axis; well, you can find hints in the “Denoising Diffusion Probabilistic Models” (Ho et al., 2020) paper we covered earlier.

140. In the experiments section of the DDPM paper, there’s a figure discussing progressive lossy compression.

141. To clarify, ‘distortion’ refers to the root mean squared error between the generated sample and the ground truth image, while ‘rate’ denotes the cumulative number of bits received up to time  $t$ .

142. Confusing, right? Let’s take a look at the algorithm to understand what this experiment is doing.

143. In this experiment, the ‘sending algorithm’ gradually sends the image from time  $t = T-1$  to  $t = 1$  as noise is progressively removed. The ‘receiving algorithm’ receives this and predicts  $\mathbf{x}_0$  at time  $t$  using the equation below.

$$\mathbf{x}_0 \approx \hat{\mathbf{x}}_0 = \left( \mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t} \epsilon_{\theta}(\mathbf{x}_t) \right) / \sqrt{\bar{\alpha}_t}$$

144. This prediction of  $x_0$  is then used to calculate distortion. Hence, distortion shows how much the predicted sample from the model trained up to time  $t$  differs from the actual answer. On the other hand, rate shows the cumulative number of bits received up to time  $t$ , indicating how much progress has been made in receiving, or in other words, how much noise has been removed from  $t = T-1$  to  $t = 1$ .

145. With this information in mind, let's revisit the figure.

146. As the reverse process progresses from the far left, i.e., as the noise is removed, the distortion decreases. In the middle graph, as the noise diminishes, the rate increases. That's natural by definition.

147. The relationship between rate and distortion is summarized in the graph on the far right.

148. This graph essentially illustrates that as the rate increases, or in other words, as the noise is removed, the root mean squared error (distortion) between the ground truth and sample images decreases.

149. Thinking about it, a slight increase in rate at the beginning drastically reduces distortion, signifying that crucial and decisive information has been learned. Conversely, even with a significant increase in rate towards the end,

distortion changes gradually, indicating the acquisition of more detailed and smaller information.

150. Now, let's go back to the diagram from 138.

151. The graph we just saw on DDPM lossy compression is exactly the same.

152. The only difference is that they've labeled it semantic compression in the early stages and perceptual compression in the later stages. What was earlier described as 'important and decisive information' in 149 is now referred to as 'semantic' here.

153. In the realm of semantic compression, the model learns the universal semantic information of the entire image that decides whether the image

becomes a cat or a person, exactly as it sounds. Naturally, as it determines the overall meaning (semantic) of the image, even slight changes would significantly alter the correct answer.

154. Conversely, in the later stages, this image expresses ‘perceptual’ as the ‘more detailed and smaller information.’ In the realm of Perceptual Compression, it basically learns specific high-frequency details like what color the cat’s eyes are or what kind of patterns exist. Even if there’s a slight change, it would mostly affect details such as a black-eyed cat versus a blue-eyed one without impacting the overall RMSE significantly.

155. Going from left to right is the reverse process step direction, which means compression progresses from right to left.

156. Without confusion, just think of it this way: learning goes in the direction of determining whether it’s a cat or a person, then deciding the color of the cat’s eyes, while compression works in the opposite direction.

157. So, ultimately, what is this image trying to convey?

158. Looking at the graph, a significant decrease in distortion occurs in the low-rate region, while the remaining bits do not contribute much to reducing distortion. **This implies that most bits are used for predicting high-frequency details that don't significantly impact the distortion.**

159. “While DMs allow to suppress this semantically meaningless information by minimizing the responsible loss term, **gradients (during training) and the neural network backbone (training and inference) still need to be evaluated on all pixels**, leading to superfluous computations and unnecessarily expensive optimization and inference.”

160. DDPM also made a similar observation.

161. “The distortion decreases steeply in the low-rate region of the rate-distortion plot, **indicating that the majority of the bits are indeed allocated**

**to imperceptible distortions.”**

**162. Despite the verbosity, the point is simple.**

**163. Performing diffusion in pixel space is wasteful.**

164. Hence, we aim to find a new space that preserves problematic perceptual information while being less computationally expensive, thus planning to train a diffusion model in this space. This way, we can reduce computational demands for high-frequency details without losing fine information.

165. This search for a perceptually equivalent but computationally suitable space is precisely what the autoencoder is used for. Let's delve into its structure.



166. First off, taking an image  $x$  with a shape of  $H \times W \times 3$  located in the RGB space as input, the encoder  $E$  encodes  $x$  into a latent representation  $z = E(x)$ . Here, the shape of the latent  $z$  is  $h \times w \times c$ .

167. Consequently, the original image is downsampled by a factor of  $H / h$  and  $W / w$ .

168. Then, the decoder  $D$  takes this latent  $z$  and reconstructs the original image.

169. Thus, this step of perceptual image compression using the autoencoder follows the basic principle of learning image features by compressing the input and then reconstructing it to its original state.

170. Through this process, the image  $x$  in the RGB space is transformed into the latent space represented by latent  $z$ .

171. **This is what the red box in the above figure represents: training the encoder and decoder.** It involves sending the image  $x$  through the encoder to the latent space and learning how to send the latent  $z$  through the decoder to the pixel space.

172. “With our trained perceptual compression models consisting of  $E$  and  $D$ , we now have access to an efficient, low-dimensional latent space in which high-frequency, imperceptible details are abstracted away. **Compared to the high-dimensional pixel space, this space is more suitable for likelihood-based generative models, as they can now (i) focus on the important, semantic bits of the data and (ii) train in a lower dimensional, computationally much more efficient space.**”

173. Now, instead of the pixel space, we need to perform diffusion in this obtained lower-dimensional, computationally efficient latent space to train the generative model.

174. **This corresponds to the green box in the figure above.** We follow the basic structure of the diffusion model but slightly modify the loss term using the equation below to proceed with the training.

175. However, there is an issue.

176. We don't just want to generate any image; rather, we want to create an image that fits specific text prompts or conditions we desire.

177. In other words, conditioning is necessary. **It's depicted in the gray box in the figure.**

178. To introduce the condition, our paper employs domain-specific encoder  $\tau_{\theta}$  and a cross-attention layer. If we designate input related to the text prompt as  $y$ , then  $y$  passes through  $\tau_{\theta}$  to encode into an intermediate representation  $\tau_{\theta}(y)$ .

179. Now, this value interfaces with the diffusion U-net through the cross-attention layer marked in orange in the figure. The cross-attention layer employs a well-known attention formula.

180. It might be familiar to most, but just in case, let's touch on it briefly.

181. Attention uses query, key, and value to calculate the attention score, indicating the relevance among tokens. To simplify, consider an example.

182. Suppose we want to translate the sentence "I am hungry" into "나는 배가 고프다" in Korean. If we want to know which English word relates most closely to "고프다" (hungry), our query would be "고프다."

183. Then, 'I,' 'am,' and 'hungry' become keys, and the similarity between each key and 'I,' 'am,' and 'hungry' is computed, multiplying these similarity scores by 'I,' 'am,' and 'hungry' (value), yielding the final attention score.

184. In essence, the key and value represent the same token, and the value multiplied by the similarity between the key and the query is the attention

score.

185. Looking back at the equation, we can observe suitable transpositions to align Q, K, and V for multiplication. Dividing by the square root of d prevents larger dot products due to longer sentences, while applying softmax ensures dominance of larger values, helping avoid excessive importance of specific tokens. These are just for reference, not crucial details.

186. Anyway, coming back to stable diffusion, we aim to employ this attention to associate the condition with the diffusion model structure. This way, it's not just random generation but 'prompt-adapted generation.' Hence, we set up Query, Key, and Value as follows.

187. As mentioned in Num 152,  $\tau_{\theta}$  serves as a domain-specific encoder. Whether a text prompt or a different form of prompt is provided,  $\tau_{\theta}$  processes it suitably to encode into a vector representation  $\tau_{\theta}(y)$ . Therefore,  $\tau_{\theta}(y)$  in Key and Value represents the encoded vector of prompt y.

188. The  $\phi_i$  in Query refers to the flattened intermediate representation of epsilon predicted by the U-net backbone of the diffusion model. In other words,  $\phi_i$  containing  $z_t$  indicates the latent  $z_t$  at time t, which the U-net uses to predict epsilon.

189. The superscript  $i$  attached to all  $W$  matrices appears to indicate the number of the attention layer.

190. To summarize, the condition encoded through  $\tau_\theta$  enters as Key and Value, intertwining with the  $\epsilon$  of the diffusion model through the **cross-attention**, ensuring that the generated samples of the diffusion model become closer to the input prompt.

191. To simultaneously train the existing  $\epsilon$  of the diffusion model and  $\tau_\theta$ , slightly modifying the equation from 174 will suffice.

192. Utilizing this loss (similar to the original diffusion model), we can train a process that, step by step, denoises from a complete noise  $z_T$  to generate a complete sample  $z_0$ .

193. Now, using this obtained latent space  $z$ , passing it through the previously trained decoder  $D$  allows us to convert it back to the pixel space and generate the final RGB image.

**194. This constitutes the complete structure of stable diffusion.**

195. In the paper's experiments section, there are performance experiments on various tasks such as image generation, conditional image generation, super-resolution, inpainting, and more. If interested, it would be beneficial to refer to it and read further.

196. I recall being astonished by many remarkable images even when skimming through papers in the past.

197. While writing this, I've come to realize that maybe I should explore some interesting papers in the field of image generation. With recent advancements in tools and significantly improved performance, I might occasionally follow up on this.

**This concludes the review, with the current content covering up to number 197.** I may add or edit information in the future to provide updates. This post is primarily for my personal understanding of the topic, and I kindly acknowledge that there may be errors or inaccuracies present. If you come across any, please feel free to bring them to my attention.

**Thank you for taking the time to read this, and congratulations 🎉 🎉 !**



- Generative Ai
- Deep Learning
- AI
- Computer Vision
- Diffusion Models




Written by Aria Lee

202 Followers

Follow




More from Aria Lee

 Aria Lee

## NumByNum :: 3D Gaussian Splatting for Real-Time Radiance...

This review of “3D Gaussian Splatting for Real-Time Radiance Field Rendering (Kerbl e...


Oct 10, 2023  163  3

 Aria Lee

## NumByNum :: ORPO—Monolithic Optimization without Reference...

This review of “ORPO—Monolithic Optimization without Reference Model (Hon...


May 1  18  1

 Aria Lee

## NumByNum :: Denoising Diffusion Probabilistic Models Reviewed

This review of “Denoising Diffusion Probabilistic Models (Ho et al., 2020)” begin...

Nov 2, 2023  27

 Aria Lee

## NumByNum :: Understanding Neural Fields & Neural Fields in...

This review of “Understanding Neural Fields & Neural Fields in Visual Computing and...

Jan 15  88  1



See all from Aria Lee

Recommended from Medium



RickyYang in The Deep Hub

A comprehensive guide to Diffusion Model-1(DDPM)

Training process of Diffusion Model



Mar 9



24



Hugman Sangkeun Jung

Variants of GAN: CGAN, Pix2Pix, CycleGAN

Explore GAN variations: CGAN, Pix2Pix, and CycleGAN. Learn their structures and...



Jun 4



1

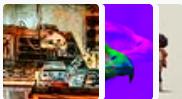


Lists



Generative AI Recommended Reading

52 stories · 1329 saves



What is ChatGPT?

9 stories · 429 saves



The New Chatbots: ChatGPT, Bard, and Beyond

12 stories · 454 saves



Natural Language Processing

1670 stories · 1251 saves



Filipa Kinomoto in Kinomoto.Mag AI

## Introducing Stable Diffusion

Stable Diffusion AI has been creating quite a buzz lately, and it's no wonder why. This...



Apr 10



51



Wei Mao in Bootcamp

## Seamless outfit swapping in Stable Diffusion ComfyUI

IP-Adapter V2 + FaceDetailer (DeepFashion)



May 12



20

Edmond Yip  in StableDiffusion

## Mastering SDXL Prompts ( 1 ) Advanced Guide for Lens...

Combining different lens distances, angles and scenes forms compositions



Jun 13



201



1



Abhishek Kumar Pandey

## Deep Diffusion Implicit Model(DDIM)

Easy:

Jun 7

[See more recommendations](#)