The Open University of Israel
Department of Mathematics and Computer Science

An Overview of Deep Learning Techniques for Image and Video Generation

Final paper submitted as partial fulfillment of the requirements
towards an M.Sc. degree in Computer Science
The Open University of Israel
Department of Mathematics and Computer Science

By
Shlomi Domnenko

Prepared under the supervision of **Dr. Mireille Avigal**

May 2025

Contents

1 Abstract

This research paper examines recent progress in image and video synthesis using machine learning (ML). While image synthesis has reached a considerable degree of maturity with the development of sophisticated deep learning models, video synthesis continues to present significant research challenges.

Deep generative models, particularly prominent models like DALL-E [67], Sora [88], Midjourney [50], have gained significant attention due to their ability to produce high-resolution and creative images and videos. In this paper we will focus on 3 image synthesis models and 3 video synthesis models that had a significant contribution to the advancement of the domain. In image synthesis we will focus on VQ-GAN [21], Stable Diffusion [71] and Imagen [75]. In video synthesis we will focus on Video-LDM [8], Stable Video Diffusion (SVD) [9] and Make-a-Video [82].

This paper surveys the recent advancements in ML techniques for image and video generation. We explore the fundamental concepts underlying these techniques, focusing on how they learn to create realistic and compelling visual content. We place particular emphasis on the video generation domain, recognizing it is immense potential and future applications.

The paper delves into various image generation models, including Variational Autoencoders (VAEs) [44], Generative Adversarial Networks (GANs) [24], and Diffusion Models (DMs) [31]. We discuss their working principles, strengths, and limitations. Additionally, we explore the use of conditioning to control the output, temporal and spatial cohesion in video synthesis, dataset preparation and learning optimization. Finally, we explore video synthesis techniques that build upon these image generation models, highlighting their unique challenges and interesting points.

We conclude that diffusion models are preferred over other baseline models, such as VAE, GAN and transformer-based models, due to their high-performance in terms of image quality and inference speed. However, with sufficient large dataset and compute resources, transformer-based generative models can outperform diffusion based models due to their ability to scale to large amounts of data. Finally, the video synthesis domain requires significant computational resources; thus, further research is needed to advance this field in response to growing computational demands.

2 Introduction

The field of image and video synthesis is large and continuously expanding. While image generation has come a long way, recent advancements in video synthesis emerged recently, highlighted by the significant breakthrough of OpenAI's Sora model [88]. A solid foundation in image synthesis techniques is essential to fully comprehend the methodologies and techniques involved in video synthesis.

In this work we will review two main models that are used as a basis in image and video generation models: Generative Adversarial Networks (GANs) [24] 5, and Diffusion Probabilistic Models (DPMs) [83] [31], 7.2.

In generative models we are given a set of data points (e.g. images) and our goal is to create a new sample (new image) from this dataset. That is, we don't want to randomly select an image from the dataset, but rather we want to create a new image that does not appear in the dataset, but is similar to it. The key word is "similar" - mathematically speaking, we are talking about predicting the probability function of the dataset. That is, the model needs to learn the underlying distribution of the dataset and create new samples that represent the same distribution. Generative models provide an efficient method for analyzing and understanding unlabeled data in unsupervised learning.

Advanced image synthesis models like DALL-E [67] and Stable Diffusion [71] represent a significant milestone in the field. While recent models may not offer groundbreaking advancements, the landscape of long video synthesis remains relatively unexplored. Video synthesis presents unique challenges compared to image synthesis, primarily due to the introduction of the temporal dimension.

New models for generating images and videos are developed by building upon existing models, enhancing them through innovative techniques and adjustments. In this work we explore the VQ-GAN model [21] 6 which combines the GAN model [24] 5 with the VQ-VAE model [95] 4 and the Transformer model [97] (appendix E) together. Notably, VQ-GAN generates images based on textual descriptions, converting text inputs into visual outputs that reflect the text's description. The VQ-VAE model, derived from the VAE model [44] 3 and employing vector quantization technique [95] 4.1, is itself an advancement of the Autoencoder model [78] 3. In short, understanding these models and their inner workings requires prior familiarity with their foundations and previous works.

Another example is the Variational Autoencoder (VAE) [44] 3 model, which is used as a basis in VQ-VAE model.

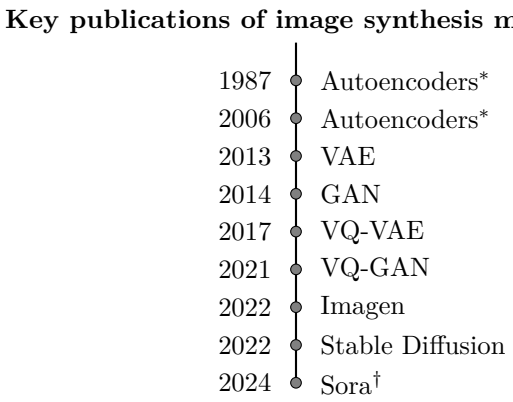


Figure 1: Chronology of key image generation models publications *The earliest mention of autoencoders appears in an 1986 publication [73], however the model was not widely used until 2006 when Hinton published a paper that uses autoencoders for dimensionality reduction [30] which sparked interest in the model again.†Although Sora [88] is not specifically an image generation model, it is considered a significant advancement in video synthesis field, which largely relies on image generation.

2.1 Mathematical Formulation of Generative Models

Mathematically speaking, let x be a random variable representing a single data point (e.g. an image) of a dataset $x_1, x_2, ..., x_n$. And let $p(x)$ denote the true probability density function (PDF) of the dataset, then our objective (as in generative modeling) is to learn a function $q(x;\theta)$ that approximates the true data distribution $p(x)$ (where θ is the model's parameters). The goal is to estimate $p(x)$ such that new samples $\hat{x} \in p(x)$ drawn from this distribution resemble the dataset.

Most of the time, it is infeasible to calculate directly $p(x)$ because computing the exact probability density for high-dimensional data is computationally expensive and often requires integrating over a large number of variables. This complexity leads to intractable calculations, making it difficult to directly model $p(x)$. Instead, we use various techniques to approximate it, and we denote it as: $q(x;\theta) \sim p(x)$.

2.2 Approximating the Data Distribution

In order to approximate $p(x)$ we can use a generative model, where the training objective is to learn the parameters θ of the model. The success or failure of the model to correctly approximate the dataset distribution can be evaluated using different loss functions, such as maximizing likelihood functions (Appendix A), minimizing Kullback-Leibler (KL) divergence, or using adversarial training loss (in the case of GANs).

2.3 Sampling

Once trained, the generative model can be used to generate new samples $\hat{x} \sim q(x;\theta)$. Sampling data point $\hat{x}$ will be consistent with the patterns and characteristics of the original dataset, as the model has learned to approximate the true data distribution. Each model will have different strategies to

sample. For example, in GANs a random noise vector is sampled from a normal distribution and passed through the generator to generate a new sample, and in VQ-GAN a transformer is used to generate latent codes that are then passed through the decoder to generate an image.

2.4 Evaluation metrics

Evaluation of the trained model is done using metrics such as sample quality, diversity (variety in generated samples), and coverage (how well the model covers the data distribution). As we will find later, one of the main problems of the GAN model is called 'mode collapse' 5.2 which causes instability of the model during training, which is manifested in large fluctuations in the loss function or in the fact that the generator fails to converge to an optimal solution that represents the entire distribution of training data. In other words, the model's output isn't diverse enough and only focuses on specific modes of the data distribution (e.g. generating only one type of image, like a cat, whereas the dataset is a collection of all animals).

Inception Score. One of the most common metrics used to evaluate the quality of generated samples is the Inception Score (IS) [77]. The IS metric measures the quality and diversity of the generated samples. A good generative model should not only produce images that look visually realistic but also capture the underlying statistical properties of the dataset. A high IS score indicates that the generated samples are both realistic and diverse. IS uses pre-trained Inception V3 model [90] to extract features and classify images to labels. The Inception V3 model is made of multiple convolutional layers and pooling layers and the last layers are fully connected layers with softmax activation function (output is probability of labels) that output the class labels (1000 classes). Two generative models are compared to each other with IS by running the same Inception V3 model, and comparing their scores relatively. The IS is calculated by first computing the conditional entropy of the generated samples given the class label:

$$p(y) = \frac{1}{N}\sum_{i=1}^{N} p(y|x_i)$$

(where y is the class label and x is an image, $p(y|x)$ is the conditional probability of the label y given image x) (i.e., the diversity of generated samples) and then computing the **KL divergence** between the marginal distribution of the generated samples and the conditional distribution:

$$D_{\mathrm{KL}}(p(y|x)\|p(y)) = \sum_{y} p(y|x)\log\frac{p(y|x)}{p(y)}$$

(where $p(y)$ is the marginal probability of the label across the set of generated images, and D_{KL} is the Kullback-Leibler (KL) divergence between the conditional label distribution $p(y|x)$ and marginal label distribution $p(y)$). Finally, we get the IS score as:

$$\mathrm{IS} = \exp\left(\frac{1}{N}\sum_{i=1}^{N} D_{\mathrm{KL}}(p(y|x_i)\|p(y))\right) = \exp(\mathbb{E}_{x\sim p_g}\left[D_{\mathrm{KL}}\left(p(y|x)\|p(y)\right)\right])$$

Where p_g is the distribution of the generated images. The IS score is the exponential of the sum of these two terms. A high IS score indicates that the generated samples are both realistic and diverse because $p(y|x)$ should be sharp (i.e. indicating generated samples are high quality), and $p(y)$ should be uniform (i.e. generated samples are diverse).

FID Score. Frechet Inception Distance (FID) score [29] is an improvement on the Inception Score and is more recent. The FID score also measures the similarity between the generated samples and the real data distribution, and the diversity. FID aims to capture this by comparing the feature distributions, not just individual image similarity. The lower the FID score, the better the model is at generating samples that resemble the real data distribution. FID leverages a pre-trained image classification model (which is also typically Inception V3 model), to extract high-level features from both the generated images and the dataset. These features capture the essential characteristics of the images, like shapes, textures, and object relationships. Then, it is possible to compare generative models based on the similarity of these features relatively to the same dataset, compared to just looking at the labels in Inception Score. The FID score is calculated as:

$$\text{FID} = ||\mu_x - \mu_g||^2 + Tr(\Sigma_x + \Sigma_g - 2(\Sigma_x\Sigma_g)^{1/2}) \tag{1}$$

where μ_x and Σ_x are the mean and covariance of the feature distribution of the dataset, and μ_g and Σ_g are the mean and covariance of the feature distribution of the generated images. The FID score is the Euclidean distance between the means and the trace (matrix operation) of the covariance matrices. A lower FID score indicates that the generated samples are more similar to the real data distribution.

3 Variational Autoencoder

Variational Autoencoder (VAE) [44] is a generative model used to learn the underlying distribution of data and generate new samples (similar to the dataset). The model consists of 3 main components: an **encoder**, **latent space** (sometimes called 'code vectors' or 'bottleneck layer') and a **decoder**. The main idea behind VAE is to use the autoencoder model [78] [4] (figure 2) to compress large dimensional space (in our case, images) into smaller, lower dimension vectors that represent the underlying hidden features. These code vectors are then fed into a decoder network which reconstructs the image (maps latent vectors to high-dimensional space).

Compared to the AE, VAEs gives significant control over how we want to model our latent distribution, which in turn gives **precise control over the output samples** (see figure 3). This kind of control doesn't exist in the usual autoencoder framework.

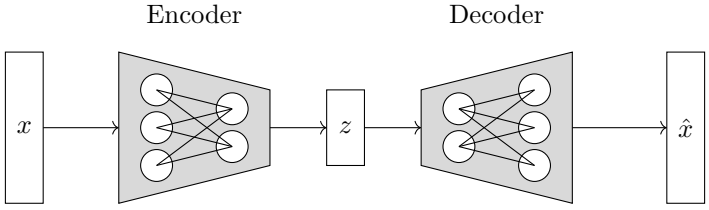
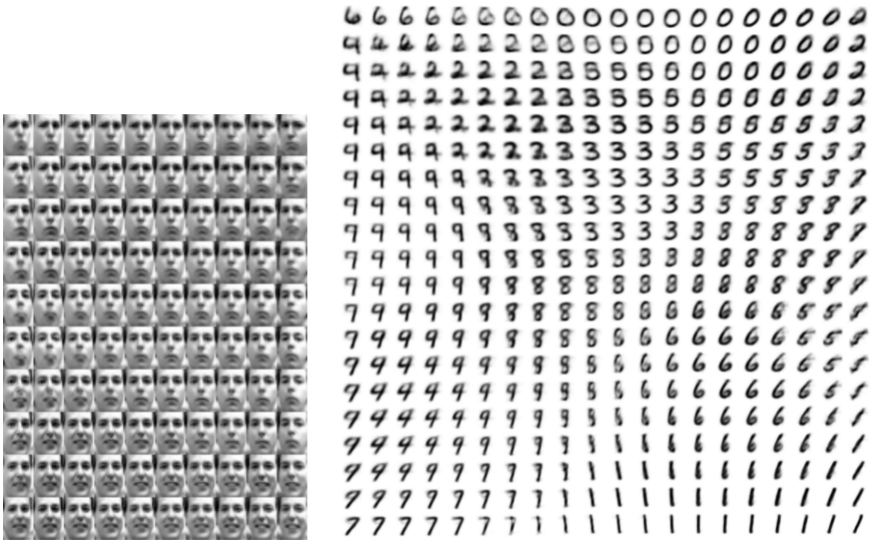


Figure 2: Autoencoder architecture.

More formally, the encoder network takes an input data point x and maps it to a latent space representation z, which is a compressed representation of x. The input is a vector, therefor an image must be flattened from 2D to 1D vector. This flattening will become an issue later in image generation, as this action removes important spatial information and hidden structures in the image. Because of this, modifications were made to the VAE model which allows the capture of spatial



(a) Learned Frey Face manifold (b) Learned MNIST manifold

Figure 3: Visualization a VAE modeled 2D latent space of learned data [44], learned on two different datasets. The variational aspect makes it possible to control the generated output. Source.

information by using a CNN (Convolutional Neural Network) [46] layers [114], max pooling layers, and other different types of spatial layers. After compression, the latent vector z is then passed onto the decoder for reconstruction.

The reconstruction is learned by a reconstruction loss function, usually **mean squared error (MSE)** loss function:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2}$$

This loss is used to ensure that the generated images closely resemble the original input images by comparing the **distance between pixels**, which motivates the model to reconstruct the image to resemble the input image.

During training, both the encoder and decoder are learned with backpropagation, but during inference, only the decoder is used.

The code vectors learned by AE are **deterministic mapping** from input to code vectors; e.g. the code vectors of images of cats are scattered throughout the entire latent space, whereas in VAE, they are clustered together.

The latent space in AEs is irregular and discontinuous which makes **interpolation in the latent space** difficult. VAE solves this problem: it regularizes the latent space by enforcing a prior distribution. This regularization leads to a smooth and continuous latent space (see figure 4), which allows the model to interpolate between the latent space smoothly, thus creating similar new images with different variations. In other words, VAE is **probabilistic model instead of discrete mapper**, like AE.

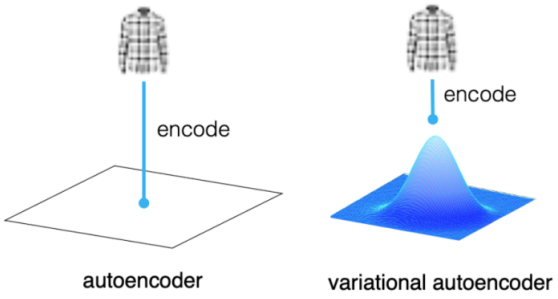


Figure 4: Mapping an input image to code vector (left) in AE and mapping an input image to a distribution (right) in VAE [1]. Source.

At the heart of VAE lies the concept of **latent variables**. Latent variables are hidden, unobserved variables that the model infers from the observed data (dataset). Latent variable models, such as VAE, take indirect approach to describing a probability distribution $p(x)$ over multidimensional variable x. Instead of directly writing the expression for $p(x)$, they model a joint distribution $p(x|z)$ of the data x and an unobserved hidden latent variable z.

The variational aspect of VAE refers to the use of **variational inference (VI)**, which is used to approximate complex posterior distributions by transforming the problem into optimization problem:

$$p(z|x) = \frac{p(x|z) \cdot p(z)}{\int p(x|z) \cdot p(z) dz} \tag{3}$$

The denominator in eq. 3 is **intractable** because it involves integrating over all possible values of z, and z is often relatively high-dimensional, and it's infeasible to evaluate exactly. Which is why VI is used, which approximates the true posterior distribution $p(z|x)$ with simpler, tractable distribution $q_\phi(z|x)$, parameterized by ϕ.

Inference: to generate an image, first we sample a latent variable z from prior distribution $p(z)$, which is typically standard normal distribution $\mathcal{N}(0, 1)$. Then z is passed to the decoder and image x is generated from the conditional distribution $p_\theta(x|z)$.

3.1 The Reparameterization Trick

To enable backpropagation through the sampling process, VAEs use the reparameterization trick. This trick involves expressing the sampled latent variables z as a deterministic function of the encoder's output and some random noise. **Without this technique, backpropagation would not be possible**, because sampling from a distribution is a stochastic process (involves randomness) and is a non-differentiable operation and the gradients can't be computed with respect to the parameters of the encoder.

To make the sampling operation differentiable and thus allow gradients to flow through the network, the reparameterization trick is used. Specifically, if μ and σ are the mean and standard deviation vectors outputted by the encoder, we can write:

$$z = \mu + \sigma \cdot \epsilon$$

where $\epsilon \sim \mathcal{N}(0, 1)$ is a standard normal random variable. This ϵ **will not change throughout the training regime**: it's sampled once and fixed in place.

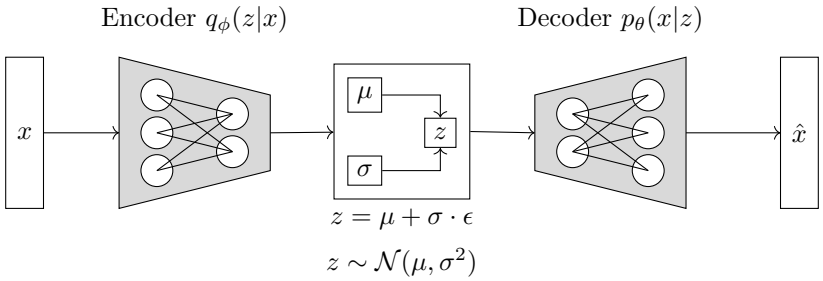


Figure 5: Variational Autoencoder architecture.

The VAE architecture is shown in figure 5.

3.2 Training

The VAE optimizes the **Evidence Lower Bound (ELBO)** to ensure that the approximate posterior $q_\phi(z|x)$ is close to the true posterior $p(z|x)$ (we want to maximize it):

$$\mathcal{L}(\theta, \phi; x, z) = \text{ELBO} = \mathbb{E}_{q_\phi(z|x)} \left[\log p_\theta(x|z)\right] - D_{\text{KL}}(q_\phi(z|x) \| p(z)) \tag{4}$$

where the first term is the **reconstruction loss** and the second term is the **KL divergence loss** (which measure how much the approximate posterior $q_\phi(z|x)$ diverges from the prior $p(z)$).

4 VQ-VAE

Vector Quantized Variational Autoencoder (VQ-VAE) [95] is a generative model based on VAE 3 with the addition of **vector quantization (VQ)** (section 4.1) technique.

We can see some generated samples created with VQ-VAE in figure 6.

4.1 Vector Quantization

Vector quantization (VQ) is a technique used to discretize continuous data. It allows the representation of a set of vectors using smaller set of representative vectors called **a codebook**. In a continuous latent space, the amount of possibilities for a value in the hidden space is infinite, which makes it **difficult for the model to learn the hidden space** efficiently.

We can see in figure 7 an example of vector quantization in a 2D latent space. The red dots represent the codebook vectors, and the grey dots represent the embeddings of the continuous latent space. We cluster all the embeddings into the closest codebook vector, and the codebook vectors are the centroids of the clusters.

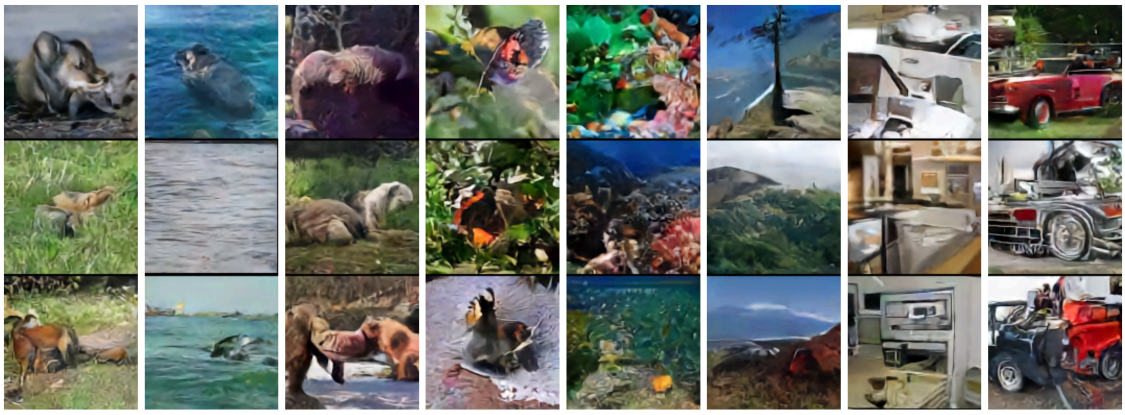


Figure 6: Samples generated by a VQ-VAE model [95], trained on ImageNet dataset [95].

In the context of VQ-VAE, the continuous latent space z is mapped into discrete codes vectors, and learning becomes more efficient because there is a fixed and smaller number of possible values in the latent space.

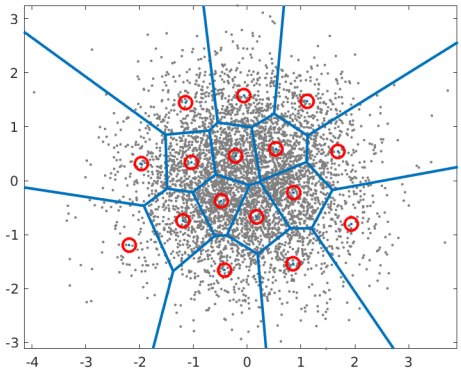


Figure 7: Vector quantization: discrete clustering of the 2D latent space. *Grey dots*: embeddings of the continuous latent space. *Red dots*: the codebook vectors. In this case, the codebook size is 16 (16 clusters) [98].

After the input passes through the encoder, the embeddings are replaced by the **closest vector from the codebook** (by minimizing distance between vectors, usually Euclidean distance).

The input vectors are partitioned into clusters and each cluster is represented by a single codebook vector (see eq. 5):

$$q(z = k|x) = \begin{cases} 1 & \text{if for } k = \text{argmin}_j \|z_e(x) - e_j\|_2 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Let K be the size of the discrete latent space and D the dimensionality of each latent embedding vector e_i. There are K embedding vectors: $e_i \in \mathbb{R}^D, i \in 1, 2, ..., K$. The model takes an input x (in the VQ-VAE paper 4 it was images) and is passed through an encoder, producing $z_e(x)$. The discrete latent variables z are then calculated by a nearest neighbor look-up: $\text{argmin}_j \|z_e(x) - e_j\|_2$ which finds the index k of the embedding vector e_j closest to $z_e(x)$.

4.2 Architecture

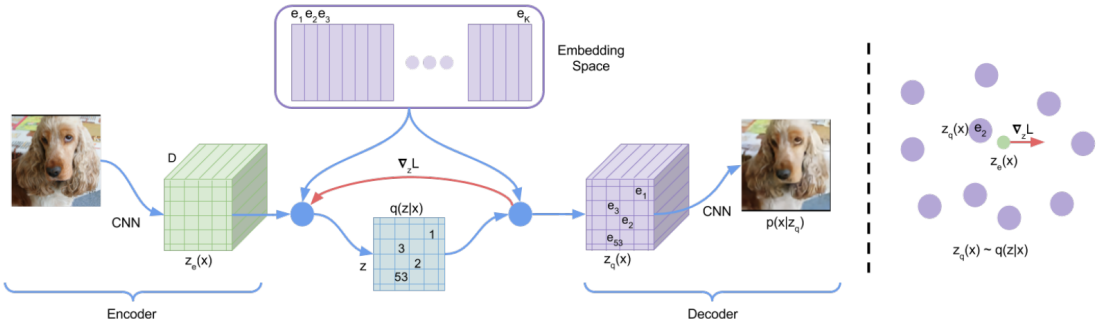


Figure 8: VQ-VAE architecture [95]. *Purple*: the code vectors. *Green*: the embeddings before quantization. *Blue*: the codebook indices (one-hot vectors). *Right*: the 'snapping' operation which replaces the embeddings with the closest codebook vector [95].

The architecture of VQ-VAE is shown in figure 8. The model is similar in structure to VAE but with the addition of vector quantization component and the codebook embeddings.

The encoder first maps the input x to a continuous latent space z_e using CNN layers. The embeddings z_e are then quantized to the nearest code vector in the codebook, resulting in the quantized embeddings z_q. These embeddings are passed to the decoder to reconstruct the image.

The quantization operation is not differentiable, that is why a straight-through estimator (STE) is used (see STE 4.4) to allow backpropagation.

4.3 Training

The **codebook is learned** by minimizing the loss function:

$$\mathcal{L}_{\text{VQ}} = ||\text{sg}[z_e] - z_q||_2^2 + \beta||\text{sg}[z_q] - z_e||_2^2 \tag{6}$$

where z_e is the encoder output, z_q is the quantized output, sg[·] is the **stop gradient operation** (which prevents gradients from flowing through the quantization operation), and β is a hyperparameter that controls the weighting of the two terms in the loss function.

The first term in the loss function is the **quantization loss**, which measures the distance between the encoder output and the quantized output.

The second term is the **commitment loss**, which measures the distance between the quantized output and the encoder output. The commitment loss encourages the model to use the codebook, and prevents the model from ignoring the quantization operation.

$$z_q(x) = e_k, \text{ where } k = \text{argmin}_j ||z_e(x) - e_j||_2 \tag{7}$$

Finding the nearest code vector in the codebook is given by equation 7.

The **loss function of the VQ-VAE** is given by:

$$\mathcal{L}_{\text{VQ-VAE}} = \underbrace{\log\, p(x|z_q(x))}_{\text{recon loss}} + \underbrace{\|\text{sg}[z_e(x)] - e\|_2^2}_{\text{quant loss}} + \underbrace{\beta\|z_e(x) - \text{sg}[e]\|_2^2}_{\text{commit loss}} \tag{8}$$

The decoder optimizes the first loss term only (reconstruction loss), the encoder optimizes the first and last loss (commitment loss) terms, and the embeddings are optimized in the middle term (quantization loss).

4.4 Straight Through Estimator (STE)

Snapping embeddings to the nearest codebook vectors operation is not differentiable, which means the gradients will always be 0 at the backpropagation [100].

The STE is a technique used to **approximate the gradients of a non-differentiable function** by passing the gradients of the function through the function itself (in other words, they skip the quantization operation in the backward pass):

In figure 8 we can see the STE with the red arrow, which copy the gradients from the decoder to the encoder (and skips the quantization process, in the middle).

4.5 Video Generation Experiment

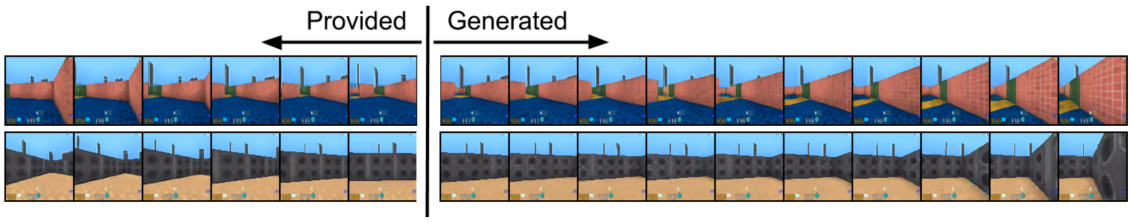


Figure 9: VQ-VAE video generation experiment [95]. *Top*: the "move forward" action. *Bottom*: the "move right" action [95].

The researchers used the VQ-VAE model to generate videos conditioned on actions (video game controls). In figure 9 we can see the results of the video generation experiment. Given 6 frames as input, the model predicted the next 10 frames conditioned on the action. Each image is created by first generating latents and then ingress into the decoder.

5 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) [24] are deep learning models that generate new data samples, particularly excelling at synthesizing high-quality images (see figure 10). The key innovation is **adversarial training**, which improves generated image quality through a two-network training approach.

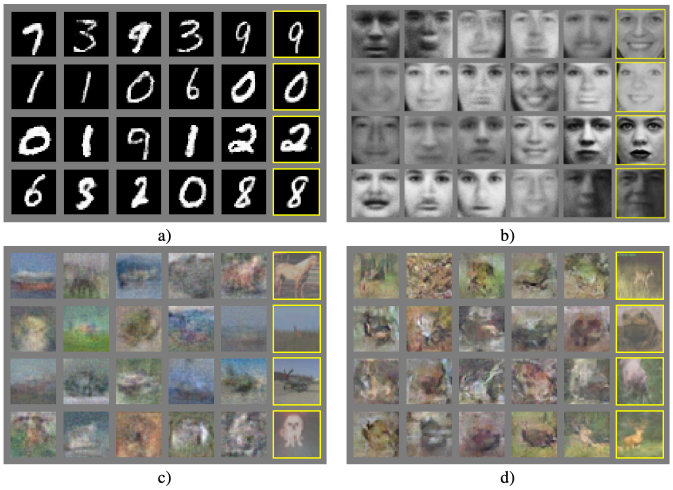


Figure 10: Some samples generated by GAN in the paper [24]. *Yellow column*: samples from the training data which closely resemble the sample to the left [24].

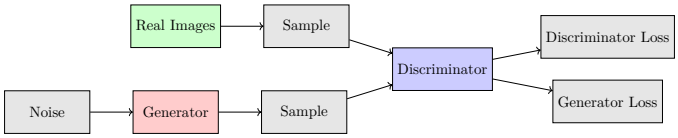


Figure 11: High-level overview of GAN architecture [24].

The model comprises two neural networks: a generator G and a discriminator D. The generator creates new samples, while the discriminator distinguishes between real and generated samples. They are trained simultaneously in a **minimax game**, where the generator aims to create indistinguishable samples and the discriminator attempts to detect generated images.

The input to the generator is a random noise vector $G(z)$ and the output is fake data, which is usually images. The input to the discriminator is either a real or fake data x (fake data generated by the generator) and the output $D(x)$ is a probability that represents the likelihood that the input x is real.

In figure 11, noise is sampled from a noise distribution (usually normal distribution) and fed to the generator. The generator outputs an image, and the discriminator decides if the image was generated by G or if it's a sample from the dataset.

Because GANs don't use variational inference like VQ-VAE, the model doesn't have control over the output image: each noise vector will generate a different image. This problem created the need for **conditional GANs**, which future papers addressed by conditioning the model on additional information (section 5.3).

5.1 Training & Adversarial loss

The GAN loss function is defined as a min-max game:

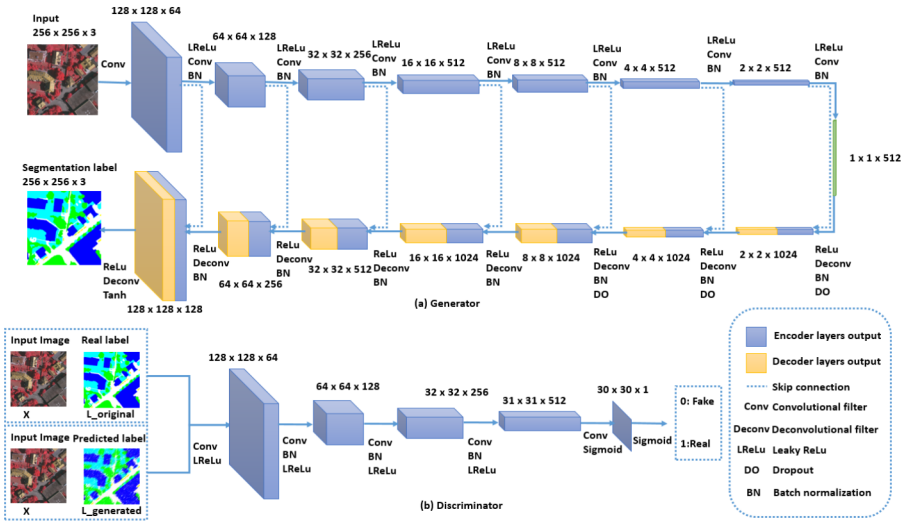


Figure 12: A more detailed architecture of GAN [7]. Both the G and D networks are made up of multiple CNN layers. Softmax in D indicates whether the image is real or fake (statistically 1 or 0) [7].

$$\min_G \max_D V(D,G) = \underbrace{\mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)]}_{\text{real}} + \underbrace{\mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]}_{\text{fake}} \tag{9}$$

In equation 9 we have two prior distributions: $x \sim p_{\text{data}}$ and $z \sim p_z(z)$, where the noise vector z is sampled from a noise distribution, and p_{data} represents the true underlying distribution of the dataset, and x is sampled from this distribution.

The loss function aims to:

1. **The first term**: Maximize the discriminator's ability to correctly classify real samples.
2. **The second term**: Minimize the generator's likelihood of producing detectable fake samples.

When we do backpropagation with respect to the generator gradients, the **first term is constant**.

Stabilizing the training: In order to balance of the training for both G and D the authors suggested using **iterative approach using mini-batch stochastic gradient decent** (see algorithm 1). Instead of fully optimizing D in each iteration, the algorithm alternates between a few steps (k steps) of optimizing the discriminator and one step of optimizing the generator.

In algorithm 1 the mini-batch size is m. We first sample m noise vectors $z_1, ..., z_m$, and we also sample mini-batch $x_1, ..., x_m$ of size m of the training data distribution $p_{\text{data}}(x)$. We then **update the discriminator** D using the formula 9 (we replaced expectation with sum since we are working in discrete finite space of mini-batch of size m, which is why we also divide by $1/m$). Similarly, we sample new noise vectors $z_1, ..., z_m$ and **update the generator** G using stochastic gradient decent, also similar to formula 9. The reason we leave out the term $\mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)]$

Algorithm 1 The training algorithm for GAN using mini-batch stochastic gradient decent [24]. The number of steps to apply to the discriminator, k, is a hyperparameter. They used $k = 1$, the least expensive option, in their experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\}$ from noise prior $p_g(\mathbf{z})$.
- Sample minibatch of m examples $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\mathbf{x})$.
- Update the discriminator by ascending it is stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\mathbf{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\mathbf{z}^{(i)}\right)\right)\right)\right].$$

end for

- Sample minibatch of m noise samples $\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\}$ from noise prior $p_g(\mathbf{z})$.
- Update the generator by descending it is stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\mathbf{z}^{(i)}\right)\right)\right).$$

end for

in the generator SGD, is because when we do gradient calculation with regard to the generator G, this term doesn't have G in it, so it becomes a constant, and we are left with the second term: $\mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$.

Why then we do a for-loop for updating the discriminator and only one update for the generator? Because the researchers wanted to balance the min-max game, and updating the discriminator faster than the prevents the generator from dominating early. However, different implementation can be used: it is possible to update both G and D at the same rate.

5.2 Mode Collapse

One of the main challenges of training GANs is mode collapse. Mode collapse occurs when the generator learns to generate only a few samples, instead of learning to generate a **diverse** set of samples. This can happen when the generator focuses only on fooling the discriminator while ignoring to diversify the output samples.

In figure 13 the blue dots are the prior $x \sim p_{\text{data}}(x)$ (the dataset ground truth) and the orange dots are the generated samples p_g by the model.

5.3 Conditional generation

Conditional generation models allows to control the output image (style, details, colors and so on). Prominent conditional GAN models include: Conditional GAN (**cGAN**) [52], **InfoGAN** [14], **CycleGAN** [113], **StyleGAN** [41] and more, each with their own unique features and capabilities.

For example, **StyleGAN** allows for the control of the style of the generated images by using a separate latent vector for each style. In **CycleGAN**, the model can learn to translate images

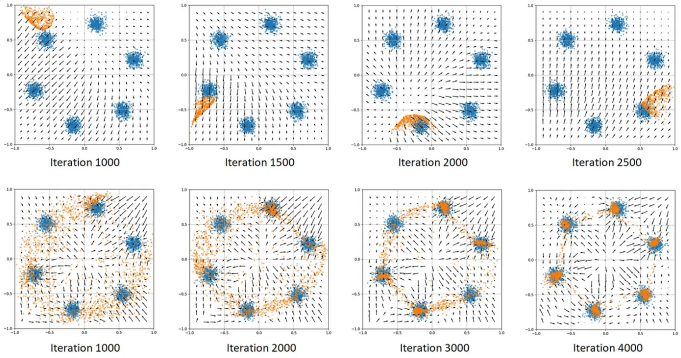


Figure 13: Mode collapse in GANs [23]. *Top row* the model fails to diversify it is output, leading it to focus on specific mode of the dataset. *Bottom row*: no mode collapse; the model successfully trained to diversify it is output [23].

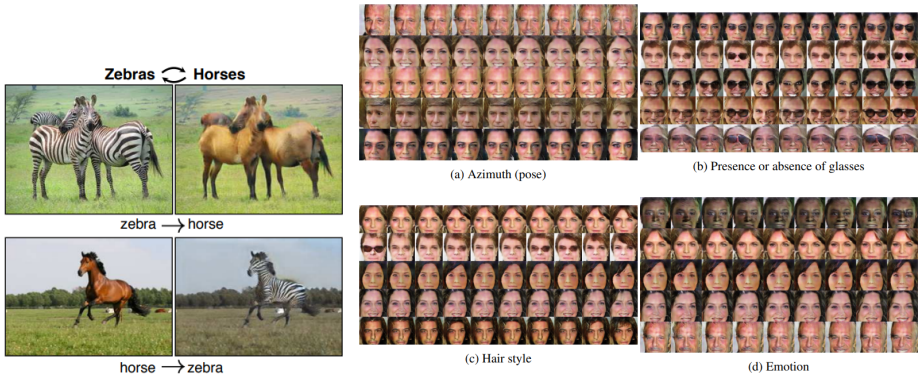


Figure 14: *Left*: **CycleGAN** can map an image from one distribution to another. *Right*: **InfoGAN** can be used to control the output (pose, glasses, hairstyle, emotion and more).

from one domain to another without paired data. And in **InfoGAN**, the model can learn the disentangled latent vectors of the data, which can be used to control the output of the generator (for example, one latent code might control the style of the human head, while another might control the hair color). See figure 14 for different examples of conditional generation for InfoGAN and CycleGAN.

6 VQ-GAN

VQ-GAN [21] is a deep learning model for high-quality image generation, combining Vector Quantized Variational Autoencoder (VQ-VAE) [95], transformers [97] (appendix E), and Generative Adversarial Network (GAN) [24] architectures. It is key innovation is representing images as perceptually rich constituents from a codebook rather than raw pixels.

VQ-GAN takes the best of both worlds: the ability to generate high-quality images from GANs

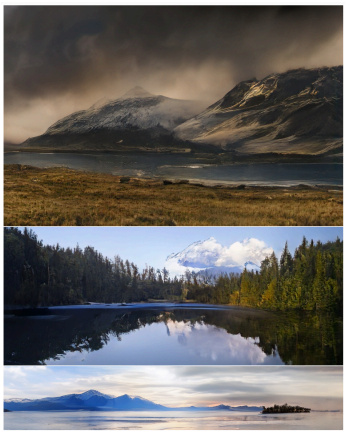


Figure 15: Samples generated by VQ-GAN with different resolutions, conditioned on semantic layouts from S-FLCKR dataset [21].

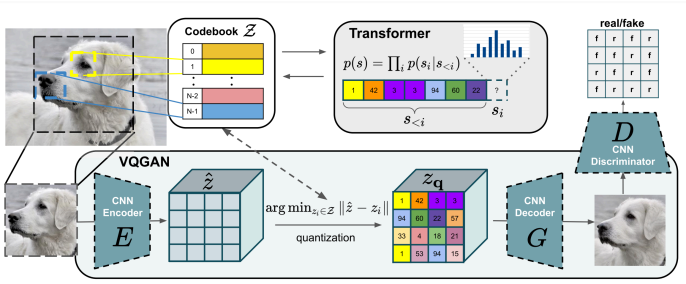


Figure 16: VQ-GAN architecture [21]. The bottom rectangular part is the VQ-VAE module with the addition of a discriminator D (right), and the top part is the autoregressive transformer network that predicts the next code vector s_i based on previous outputs $s_{<i}$ [21].

and the ability to condition the generation process from VQ-VAE using transformers.

Transformers can learn **long-range dependencies**, whereas CNNs [46] are better fit at learning **local features** and structures of images.

In figure 15 we can see samples generated by VQ-GAN with different resolutions, conditioned on semantic layouts.

In the source code of VQ-GAN the researchers used the **VGG16** [81] architecture as the backbone of the encoder and decoder networks, but they mentioned that other architectures can be used as well, depending on the generative task. In addition, the authors used the **minGPT** [61] architecture as the transformer module (more commonly known as **GPT-2**, which is based on the OpenAI's model GPT-1).

The non-differential operation of quantization is a problem we saw previously in VQ-VAE; to solve this problem the authors said that they used the **straight-through estimator** (STE) [5] to backpropagate the gradients through the quantization process, similarly to VQ-VAE.

6.1 Architecture

The architecture of VQ-GAN is shown in figure 16:

- **Input**: images $x \in X$ of size $x \in \mathbb{R}^{H \times W \times 3}$, the training images resolution is 256×256.
- **Encoder** E converts images to latent representations $\hat{z}$ of size $\hat{z} \in \mathbb{R}^{h \times w \times n_z}$, where n_z is the dimension of each codebook vector.
- **Vector Quantization** (VQ) module discretizing latent vectors $z_q \in \mathbb{R}^{h \times w \times n_z}$.
- CNN decoder G reconstructs the image $\hat{x}$ given z_q.
- **Patch-based discriminator** D distinguishes between real and fake pixel patches of size 16×16, which enhances the generator to output better quality images.
- **Inference**: the autoregressive transformer predicts the embeddings z_q and provide the decoder G the necessary information to generate a new image (compared to PixelCNN [96], which uses transformer for pixel-by-pixel prediction).

6.2 Training

The model trains in two phases:

1. Training the VQ module to learn discrete latent representations of the input data (learns the codebook).
2. Training a transformer to predict the codebook vector sequences.

The VQ-GAN model has 3 loss functions:

1. The vector quantization (VQ) loss, which trains the model to learn the codebook vectors.
2. The adversarial (GAN) loss, which trains the model to generate realistic images.
3. The transformer loss, which trains the model to predict the next code vector autoregressively.

Vector quantization loss

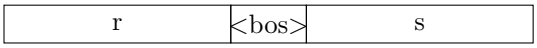
The loss function is similar to the VQ-VAE loss (equation 8) and is given by:

$$\mathcal{L}_{\text{VQ}}(E, G, \mathcal{Z}) = \underbrace{\|x - \hat{x}\|^2}_{\text{recon loss}} + \underbrace{\|\text{sg}[E(x)] - z_q\|_2^2}_{\text{quant loss}} + \underbrace{\beta \|\text{sg}[z_q] - E(x)\|_2^2}_{\text{commit loss}}$$

The loss function consists of three terms: the reconstruction loss, the quantization loss, and the commitment loss:

- The reconstruction loss is intended to train the decoder G to reconstruct latents z_q and output images of similar distribution to the dataset.

Figure 17: The conditioned input sequence given to the transformer, based on the spatial condition information c. The first sequence r are the representation of c. The middle rectangle represents a 'begin of sequence' token. And s is the sequence tokens of the input x (the indices z_q in figure 16).



- The quantization loss encourages the model to move the codebook vectors towards the encoder outputs, so to match the encoder's output distribution (gradients are calculated for z_q and not $\text{sg}[E(x)]$ because of the stop gradient operation).
- The commitment loss encourages the encoder to "commit" to outputting embeddings that are close to the codebook vectors.

However, instead of the MSE loss, the authors used a **perceptual loss** [110] §

GAN loss

The adversarial objective trains the patch-based discriminator D to try and distinguish between real and fake pixel patches of size 16×16 of an image. The loss function of the discriminator is given by:

$$\mathcal{L}_{\text{GAN}}(\{E, G, Z\}, D) = [\log D(x) + \log(1 - D(\hat{x}))]$$

The first term encourages the discriminator to output 1 (predicted real image from the dataset) because $\log D(x)$ approaches 0 when $D(x)$ approaches 1, and the second term encourages the discriminator to output 0 (predicted fake image from G) for fake images because $\log(1 - D(\hat{x}))$ approaches 0 when $D(\hat{x})$ approaches 0.

Combining both the VQ loss and the adversarial loss, the total loss function is given by:

$$\mathcal{Q}^* = \arg \min_{E,G,Z} \max_{D} \mathbb{E}_{x \sim p(x)}[\mathcal{L}_{\text{VQ}} + \lambda \mathcal{L}_{\text{GAN}}]$$

The λ hyperparameter is used to balance the contribution of adversarial loss relative to the VQ loss.

Transformer objective

The second phase of the training involves maximizing the transformer objective. The transformer learns to predict the distribution of possible next indices, which allows us to directly maximize the log-likelihood (appendix A) of the data representation:

$$\mathcal{L}_{\text{transformer}} = \mathbb{E}_{x \sim p(x)}[-\log p(x)]$$

6.3 Conditional generation

After the two-phase training is finished, the transformer is used to predict a sequence s which is sequence of indices to code vectors (each token s_i corresponds to an index in the codebook). In other words, a code vector is autoregressively predicted based on the previous tokens $s_{<i}$, which provides the necessary embeddings z_q for image synthesis:

$$p(s) = \prod_i p(s_i|s_{<i}) \tag{10}$$

Equation 10 represents the autoregressive conditional synthesis objective of the transformer.

To involve conditioning information c such as text, images, depth map, semantic layout or pose, a **new VQ-GAN model** is trained on each kind of conditioning modal*. If the conditioning information c has spatial extent (consists of spatial information like images, semantic masks...), then we learn a new VQ-GAN on this data to obtain index-based representations $r \in \{0, ..., |\mathcal{Z}_c|-1\}^{h_c \times w_c}$ (similar to $s_i \in z_q$ in the first VQ-GAN model) of c with the new codebook $\mathcal{Z}_c$. Then, this representation r is prepended to s (figure 17) as an input sequence to the transformer, which restricts the computation of the negative log-likelihood to entries $p(s_i|s_{<i}, r)$.

6.4 Sliding window attention technique for generating high-resolution images

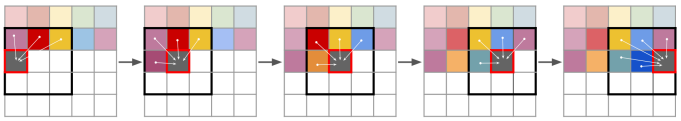


Figure 18: Sliding window attention technique proposed in the paper [21].

The problem: the attention mechanism in transformers requires quadratic computation for the number of tokens, because each token requires attention to all other tokens in the sequence (in the paper the sequence is limited to 256 tokens). Therefor they decided to use a sliding window attention mechanism to reduce the computational cost.

Sliding window attention technique: to solve this problem, they work patch-wise and crop input images to match the maximum token count in the transformer module, so the transformer works on the window context only, instead the entire image, which reduces the token count required.

In figure 18, the output image is divided into patches of 16×16 the transformer is able to autoregressively predict the next token (code vector) s_i based on the previous tokens $s_{<i}$ (the window context).

Sampling: to sample images, they also work patch-wise and the transformer autoregressively predicts the next token based on the window context (figure 18).

§Perceptual loss measures the difference between the high-level features of two images (a generated image and an image from the dataset). Usually, the high-level features are extracted by pre-trained CNNs networks and then are compared.

*Conditioning modal refers to specific type of data representation given as the conditioning signal. For instance, it can be images, text, video, etc.

7 Denoise Diffusion Probabilistic Models (DDPMs)

7.1 Diffusion Models (DMs)

Diffusion models are probabilistic generative models that map latent variables to observed data. Unlike GANs, they are easier to train and can produce high-quality images by progressively adding and removing noise. Most importantly, this model serves as a basis to other generative models.

A diffusion model consists of:

- An encoder that maps the data x through stochastic intermediate latent variables $z_1, ..., z_T$.
- And a decoder reversing the noising process

7.2 DDPMs

Denoise Diffusion Probabilistic Models (DDPMs) [31] are a class of diffusion models that are trained to denoise images by reconstructing a noised image.

There are two processes, each process takes multiple (t) steps:

- **Forward diffusion process**: Adds noise to the input image[†]. The addition of noise is the addition of random pixel values to the input image. In forward diffusion, we transform the image x (from the dataset, without noise) to a latent variable $z_0 = x$. Then we add noise to get z_1, and so on, until t steps we get z_t - which is pure noise image, distributed from the noise distribution (usually Gaussian). The forward diffusion step is denoted as $\mathbf{q(x_t|x_{t-1})}$.
- **Reverse diffusion process**: Removes noise from the input noised image. The removal of noise can be thought as adding new features to the image. In reverse diffusion, we transform the image z_t to z_{t-1} by removing noise in a single step. We repeat this process until we get the original image $x = z_0$. The reverse diffusion step is denoted as $\mathbf{p_\theta(x_{t-1}|x_t)}$.

Because the addition of noise is a known stochastic process, **all the learned parameters are in the decoder** (the generator). We don't want to learn the addition of noise, since the task is to denoise the image. DDPMs learn with a fixed inference procedure $q(x_{1:T}|x_0)$.

DDPMs are latent variable models in the form of:

$$p_\theta(x_0) = \int p_\theta(x_{0:T}) dx_{1:T}, \quad \text{where} \quad p_\theta(x_{0:T}) := p_\theta(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$

Because this **integral is intractable**, we use ELBO. The parameters of the model θ are learned to fit the data distribution $q(x_0)$ by **maximizing a variational lower bound**:

$$\max_\theta \mathbb{E}_{q(x_0)}[\log p_\theta(x_0)] \leq \max_\theta \mathbb{E}_{q(x_0,x_1,...,x_T)}[\log p_\theta(x_{0:T}) - \log q(x_{1:T}|x_0)]$$

where $q(x_{1:T}|x_0)$ is the reverse diffusion process over the latent variables.

In figure 19 we see the progressive denoising of the noised image (on the left) in steps, until we generate an image (on the right). This is the reverse diffusion process.

[†]There are multiple implementations on how to add noise to image. The simplest way is to use random values for each pixel (`torch.randn_like()`) by tensor addition, but the most common way is to use Gaussian noise which distributes the noise normally (`np.random.normal(mean, sigma, ...)`).

Figure 19: Progressive generation (left to right) of unconditional CIFAR10 dataset in DDPM [31].

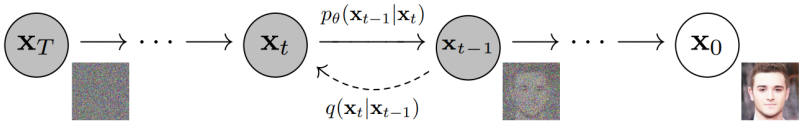


Figure 20: A graph representing the forward ($q(x_t|x_{t-1})$) and reverse ($p_\theta(x_{t-1}|x_t)$) diffusion process in DDPMs [31]. The next step (either in forward or reverse diffusion) depends conditionally on the previous steps (Markov chains) [31].

In figure 20 we see the forward and reverse diffusion process in DDPMs. The forward diffusion process q adds noise to the image, and the reverse diffusion process p removes noise from the image. The next step in the diffusion process depends conditionally on the previous steps, hence the conditional probability.

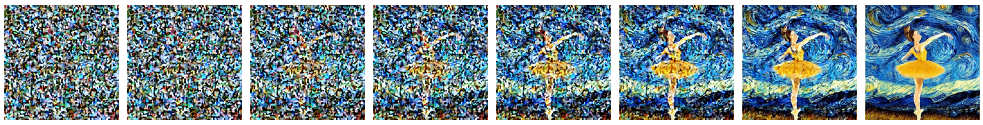


Figure 21: Progressive denoising of latents in DDPM. Images taken from the harvard university website.

And finally in figure 21 we see the progressive denoising of latents in DDPMs. We see at the beginning seemingly random noised image, however it is not completely random. It has some patterns to it, indicating this latent (image) is conditioned on something before creating it. As we progress through the steps, we see the image getting clearer and clearer, until we get the generated image.

7.3 Noise Schedulers

A noise scheduler defines how noise is added to the image at each timestep (for the forward process in DDPMs). Instead of adding noise all at once, a scheduler defines how much noise to add periodically (in steps).

Typically, we control noise schedulers with two parameters: α_t which controls how much noise is added (strength of the noise) at step t, and β_t which controls the variance of the noise added at step t (typically referred to as just 'noise schedule' or 'variance').

Figure 22: *Top*: Linear scheduler [31]; *Bottom*: cosine scheduler [55]. The linear scheduler adds noise too quickly which degrades the model's performance quickly, whereas cosine scheduler adds noise more slowly [55].

In the paper [31] the authors used linear scheduler, however OpenAI released a paper [55] that uses cosine scheduler. They have shown that a cosine scheduler performs better than a linear scheduler in terms of image generation quality (see figure 22).

A variance schedule is simply a function that defines the variance at each given timestep during the forward diffusion process. At each timestep, Gaussian noise ϵ is added to the previous latent variable:

$$x_{t+1} = \sqrt{1-\beta_t}x_t + \sqrt{\beta_t}\epsilon$$

Notice that when β increases linearly, the term $\sqrt{1-\beta}$ decreases linearly.

A noise scheduler is a function: $\beta(t) : [1, ..., T] \to \mathbb{R}$ that determines the amount of noise added at each step. It is a sequence of α_t.

A **linear scheduler** is defined as:

$$\beta_t = c \cdot t, \text{ where c is constant}$$

and a **cosine-beta scheduler** can be defined as:

$$\bar{\alpha}_t = \frac{f(t)}{f(0)}, \text{ where } f(t) = \cos^2\left(\frac{t/T + s}{1+s} \cdot \frac{\pi}{2}\right)$$

7.4 Encoder

The forward diffusion process maps a data image x through a series of intermediate variables $z_1, ..., z_T$ with the dimension as x according to the following recursion:

$$\begin{aligned}
\mathbf{z}_1 &= \sqrt{1-\beta_1} \cdot \mathbf{x} + \sqrt{\beta_1} \cdot \epsilon_1, \\
&\vdots \\
\mathbf{z}_t &= \sqrt{1-\beta_t} \cdot \mathbf{z}_{t-1} + \sqrt{\beta_t} \cdot \epsilon_t \quad \forall\, t \in \{2, \ldots, T\}
\end{aligned}$$

At each step $i \in \{1, 2, ..., t, t+1, ..., T\}$ a noise vector ϵ_i is drawn from a standard normal distribution. This equation adds random noise ϵ_i to the image $x = z_0$ and β is a schedule function ($\beta(t) : [1, ..., T] \to \mathbb{R}$) that determines the amount of noise added at each step (β is a variance schedule, which can be linear, cosine, quadratic and more).

A more formal notation for the **forward diffusion process** is:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t} \cdot x_{t-1}, \beta_t \mathbf{I})$$
$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$$

- The noise is sampled from a normal distribution $\epsilon \sim \mathcal{N}(0, 1)$.
- The latent variable x_t is sampled from a normal distribution $\mathcal{N}$ where:
  - The mean is $\sqrt{1-\beta_t} \cdot x_{t-1}$.
  - The variance is $\beta_t \mathbf{I}$.
  - The β parameter is the noise scheduler (how much noise we add at each step).

Skipping intermediate steps in forward diffusion

An interesting point the authors made is that in forward diffusion it is possible to sample x_t from any arbitrary timestep t, given the original image x_0 (without calculating all the intermediate steps):

$$\begin{aligned} q(\mathbf{x}_t|\mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}) \\ \text{where } \alpha_t &:= 1-\beta_t \text{ and } \bar{\alpha}_t := \prod_{i=1}^{t} \alpha_i \end{aligned} \tag{11}$$

Since α depends on β we can see that **there are no parameters to learn in the forward diffusion process**.

After plugging in $\alpha = 1 - \beta$ in the encoder and the **reparametrization trick** (from VAE: $\mathcal{N}(\mu, \sigma^2) = \mu + \sigma \cdot \epsilon$, where $\epsilon \sim \mathcal{N}(0, 1)$) we get:

$$\begin{aligned} q(x_t|x_{t-1}) &= \mathcal{N}\left(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t \mathbf{I}\right) \\ &= \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_t \\ &= \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}\epsilon \\ &= \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\epsilon \\ &= \sqrt{\alpha_t\alpha_{t-1}\alpha_{t-2}}x_{t-3} + \sqrt{1-\alpha_t\alpha_{t-1}\alpha_{t-2}}\epsilon \\ &= \sqrt{\alpha_t\alpha_{t-1}\cdots\alpha_1\alpha_0}x_0 + \sqrt{1-\alpha_t\alpha_{t-1}\cdots\alpha_1\alpha_0}\epsilon \\ &= \boxed{\sqrt{\bar{\alpha}}x_0 + \sqrt{1-\bar{\alpha}}\epsilon} \end{aligned}$$

as a result we get:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{12}$$

Equation 12 describes how in a single step we can get a noisier image x_t from the original image x_0 without calculating all the intermediate steps. This is a very useful property of latent diffusion models.

7.5 Decoder

The decoder removes noise from the intermediate latent variables $z_T, ..., z_1$ and reconstructs the original image $x = z_0$ using the following recursion:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)) \tag{13}$$

where θ are the learned parameters of the model. The mean μ_θ and variance Σ_θ are unknown and should be learned in the training.

Learning the variance

In the DDPM paper [31] the authors said:

> *"We also see that learning reverse process variances (by incorporating a parameterized diagonal $\Sigma_\theta(x_t)$ into the variational bound) leads to unstable training and poorer sample quality compared to fixed variances."* [31]

The OpenAI team released a paper titled "Improved denoising diffusion probabilistic models" [55] in which the authors used cosine noise (instead of linear in the original paper [31]) scheduler and also **learned the variance**, which significantly improved the model's performance.

7.6 Loss function

Diffusion models are in the form of $p_\theta(x_0) := \int p_\theta(x_{0:T}) d\mathbf{x}_{1:T}$ which is intractable, thus we use the ELBO loss function to approximate the likelihood of the data.

We seek to train the model to predict the noise ϵ_i added at each timestep. We could start with a simple loss function (**negative log likelihood**):

$$-\log(p_\theta(x_0))$$

but that's a problem, the probability of x_0 depends on all timesteps $x_0, x_1, ..., x_T$. As a solution, loss function of DDPMs is typically the **variational lower bound** of this objective:

$$-\log(p_\theta(x_0)) \leq -\log(p_\theta(x_0)) + D_{KL}(q(x_{1:T}|x_0))||p_\theta(x_{1:T}|x_0)$$

We can rewrite the KL-divergence as [‡]:

[‡]Thanks in part for the math explanation to a youtube video that explained the math in the paper.

$$
\begin{aligned}
D_{KL}(q(x_{1:T}|x_0))||p_\theta(x_{1:T}|x_0) &= \log(\frac{q(x_{1:T}|x_0)}{p_\theta(x_{1:T}|x_0)}) \\
&= \log(\frac{q(x_{1:T}|x_0)}{\frac{p_\theta(x_0|x_{1:T})p_\theta(x_{1:T})}{p_\theta(x_0)}}) \\
&= \log(\frac{q(x_{1:T}|x_0)}{\frac{p_\theta(x_{0:T})}{p_\theta(x_0)}}) \\
&= \log(\frac{q(x_{1:T}|x_0)}{\frac{p_\theta(x_{0:T})}{p_\theta(x_0)}}) \\
&= \log(\frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}) + \log(p_\theta(x_0))
\end{aligned}
$$

By reformulating the KL-divergence we arrive at the following loss objective (the terms $-\log(p_\theta(x_0))$ and $\log(p_\theta(x_0))$ cancel each other out):

$$
\begin{aligned}
-\log(p_\theta(x_0)) &\leq -\log(p_\theta(x_0)) + \log(\frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}) + \log(p_\theta(x_0)) \\
&= \log(\frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})})
\end{aligned}
$$

we finally get the **variational lower bound (ELBO)**:

$$
-\log(p_\theta(x_0)) \leq \log(\frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})})
$$

$$
\boxed{\mathcal{L} = \underbrace{\mathbb{E}[-\log p_\theta(x_0)]}_{\text{NLL}} \leq \underbrace{\mathbb{E}_q[-\log(\frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)})]}_{\text{ELBO}}}
$$

The dominator $q(x_{1:T}|x_0)$ is the forward diffusion process (adds noise) and the numerator $p_\theta(x_{0:T})$ is the reverse process.

The ratio of these two terms is the ELBO loss function and when this ratio reaches 1, it indicates that the model can undo the forward process. We also don't write q_θ because the forward process is not learned, only the reverse process is learned.

We can simplify the training objective of DDPMs as simply **noise predictor** - commonly written as ϵ-prediction:

$$
\boxed{\mathcal{L}_{\text{simple}}(\theta) = \mathbb{E}_{t,x_0,\epsilon}\left[||\epsilon - \epsilon_\theta(x_t, t)||^2\right]} \tag{14}
$$

where $\epsilon_\theta(x_t, t)$ is the noise predicted by the model, given timestep t and noisy input x_t.

Figure 23: Stable diffusion scales better compared to other models [71] (DALL-E, VQ-GAN) with less downsampling blocks ($f = 4$ instead of 16 as needed by VQ-GAN).

7.7 Training

The training of diffusion models is done by sampling a batch of images from the dataset, and then adding noise to the images in the forward diffusion process. The model is trained to remove the noise and reconstruct the original image in the reverse diffusion process. The loss function is the ELBO loss function. Usually, the model is trained using stochastic gradient descent (SGD), but more advanced optimizers like Adam can be used as well.

Algorithm 2 The training algorithm of diffusion models [31].

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on

$$\nabla_\theta \left\| \epsilon - \epsilon_\theta \left(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, t \right) \right\|^2$$

6: **until** converged

In algorithm 2 line 2, we take a sample from the dataset. In line 3, we generate random number between 1 and T uniformly. Line 4: we sample some noise. Line 5: we calculate the gradients of the loss function: we try to optimize the model's parameters θ by gradient decent. ϵ_θ is the predicted noise added at timestep t (an approximation function with 2 parameters that intends to predict ϵ from x_t) where the first parameter is the noisy image at timestep t, and the second parameter is the timestep t.

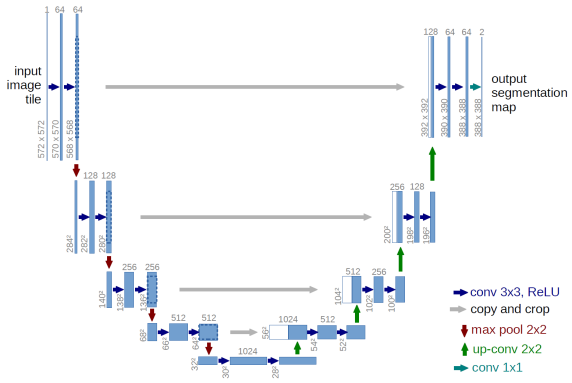


Figure 24: The U-Net architecture [72] with convolutional and deconvolutional layers & skip connections.

8 Stable Diffusion (Latent Diffusion Model)

In the Stable Diffusion paper [71] the authors suggested that computing gradients directly in DDPMs on the pixel space is inefficient, since this space is highly-dimensional and includes undesired high-frequency details. They suggest converting the input images to a lower-dimensional latent representations and then apply the diffusion processes. The authors showed that this approach **scales better** and is more compute efficient [§] compared to working in pixel space. Moreover, the authors introduced general purpose **conditioning mechanism based on cross-attention** which allows multi-modal training.

A 2021 paper released by OpenAI [19] shows that **diffusion models can outperform GANs** in terms of image fidelity by trading off diversity.

In figure 23 we see that Stable Diffusion scales better than other methods (VQ-GAN, DALL-E) while using less downsampling blocks ($f = 4$ instead of 16 as needed by VQ-GAN). This is a significant improvement in terms of compute efficiency and memory usage.

8.1 The U-Net backbone

U-Net (first introduced in 2015) [72] is a convolutional neural network (CNN) architecture that is commonly used in diffusion models. U-Net is used as a backbone for denoising the latent variables $z_1, ..., z_T$. The U-Net architecture is a symmetric encoder-decoder network with skip connections between the encoder and decoder: the skip connections help the network to learn better by minimizing the **exploding / vanishing gradient problems** [6].

In figure 24 the U-Net is shaped like a 'U' in which the input is downsampled to low spatial resolution and high feature channels and then upsampled back again. The convolution kernel size is 3x3, and they use ReLU activation function.

[§] In the Stable Diffusion paper [71] the authors showed that the model can be trained on a single GPU with 16 GB of memory on images of 256×256 resolution using CelebA-HQ dataset with 30 diffusion steps.

8.2 Sinusoidal embeddings

The diffusion process uses **timestep embeddings** which is the embeddings representing the current diffusion timestep: $t \in [0, T]$. Instead of directly using timestep scalar, we first convert it to embeddings. This way the model knows if it's in the beginning of the diffusion or at the later stages, which is essential for removing noise (because noise schedulers depend on the current timestep).

To get the embeddings, the timestep is first projected into a **sinusoidal embedding**, similar to the way positional encodings are used in transformers.

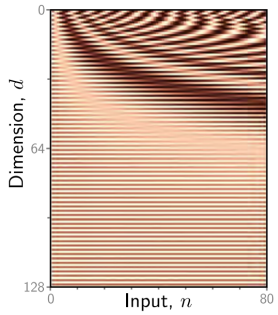


Figure 25: Sinusoidal positional embeddings used in a transformers [59]. *X-axis*: the input scalar (position in a transformer or timestep in diffusion model). *Y-axis*: the embeddings dimensions [59].

In figure 25 we can see the sinusoidal positional embeddings used in a transformer model, which is similar to timestep embeddings. The pattern is sinusoidal where lighter regions indicate lower values and darker regions indicate higher values. Each column in the X axis is a **unique embedding**. For example, the scalar '1' will have different encodings than a scalar '2', which helps distinguish the position / timestep of the input sequence.

Listing 1: Timestep embeddings in Stable Diffusion: we convert the timestep to an embedding.

```
def get_time_embedding(timestep):
    # Shape: (160,)
    freqs = torch.pow(10000, -torch.arange(start=0, end=160, dtype=torch.
        float32) / 160)
    # Shape: (1, 160)
    x = torch.tensor([timestep], dtype=torch.float32)[:, None] * freqs[None]
    # Shape: (1, 160 * 2)
    return torch.cat([torch.cos(x), torch.sin(x)], dim=-1)
```

The general formula for timestep embeddings is given in equation 15. The code snippet in listing 1 is taken from a re-implementation of Stable Diffusion.

$$\text{Enc}_i(t) = \begin{cases} \sin\left(\frac{t}{10000^{\frac{2i}{d}}}\right) & \text{if } i \text{ is even} \\ \cos\left(\frac{t}{10000^{\frac{2i}{d}}}\right) & \text{if } i \text{ is odd} \end{cases} \tag{15}$$

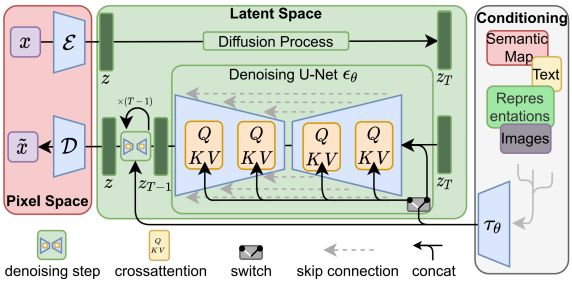


Figure 26: Stable Diffusion architecture [71].

8.3 Architecture

The high-level architecture is shown in figure 26. The stable diffusion model, which is a latent diffusion model (LDM), consists of:

- Variational autoencoder (VAE) which compresses the input images into regularized latent space. It consists of:
  - Encoder ε which converts the input images x to latent space z.
  - Decoder $\mathcal{D}$ which converts the latent vector z back to the pixel space $\tilde{x}$.
- A U-Net backbone which serves as the noise prediction network, predicts the noise needed to denoise the intermediate latent vectors $z_T, ..., z_1$ in each step. It incorporates timestep embeddings as an essential input.
- A domain specific encoder τ_θ which encodes the conditional information (text prompts, images, segmentation masks) into tokens which will be used in the cross-attention layers, or concatenated with the latent vector (the switch mechanism).

8.4 Conditioning

Cross-attention (appendix D) is used in Stable Diffusion for **multi-modal conditioning**: it guides the model to output images based on conditional information. Although this information is multi-modal (e.g. images, text, segmentation masks), the cross-attention layers in the U-Net and the switch mechanism able to process them all.

Text conditioning: if the conditioning signal is text prompt, the domain specific encoder τ_θ is a text encoder (tokenizer) which converts the text words to embeddings (tokens). In the paper [71] the authors used a **frozen CLIPTokenizer**, which is a pre-trained tokenizer trained on specific vocabulary, and special tokens (beginning of sentence, end of sentence, padding, mask tokens and more). However, digging into the source code, you will find other text encoders as well such as **BERT** [18].

Switch mechanism: In figure 26 the 'switch' in the diagram is used for different kinds of conditional information. If the conditional information is spatial (such as images, layouts, semantic masks), they use concatenation with z_T. In the case of text (not spatial), they use cross-attention layers.

We dive deeper into self-attention, multi-head attention and cross-attention in the appendix D, which are commonly used in other image and video synthesis models.

8.5 Classifier-free diffusion guidance (CFG)

Classifier-free guidance (CFG) is used in Stable Diffusion to control the influence of the conditioning signal and balance between being faithful to the conditioning signal or being diverse in the generated samples.

So far we have focused on modeling just the data distribution $p(x)$. However, we are often also interested in learning conditional distribution $p(x|y)$, which would enable us to explicitly control the data we generate through conditioning signal y.

We can add conditioning information alongside the timestep information, at each iteration (appendix B equation 8 of [31], appendix H equation 31 of [19], equation 1 in [84]):

$$p(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_{t,y})$$

where p is the reverse diffusion process that removes noise in image x_t in timestep $t \in T, T-1, ..., 0$, and y is the conditioning information. This equation shows that the reverse diffusion process depends on previous reverse diffusion steps (product of previous steps) and the initial image x_0.

When training a diffusion model to generate images based on specific conditional information, there's a risk that the model might not fully consider or even ignore these conditions, using this vanilla formulation. To address this, a technique called "guidance" is used. Guidance allows us to explicitly control **how much influence the conditions have on the generated images**, but this can sometimes lead to less variety in the results. In other words, we use weight to control how much the model should pay attention to the conditioning signal.

Conditioning a generative model can be achieved through two methods: **classifier guidance** and **classifier-free guidance**.

Classifier guidance

Classifier guidance [19] involves **training a separate model** to condition the output, and is based on score-based diffusion models [85].

The formulation of classifier guidance is given as:

$$\nabla \log p(x_t|y) = \underbrace{\nabla \log p(x_t)}_{\text{unconditional score}} + \underbrace{\gamma \nabla \log p(y|x_t)}_{\text{adversarial gradient}}$$

where γ is a hyperparameter that controls the strength of the conditioning signal in classifier guidance method.

Classifier-free guidance

In Classifier-free guidance (CFG) [32], instead of training two networks, one conditional network and an unconditional network, we train a single network, and during training, **we set the conditioning signal to zero** with some probability. This way, the network becomes a mix of conditioned and unconditioned networks, and we can take the conditioned and unconditioned output and combine

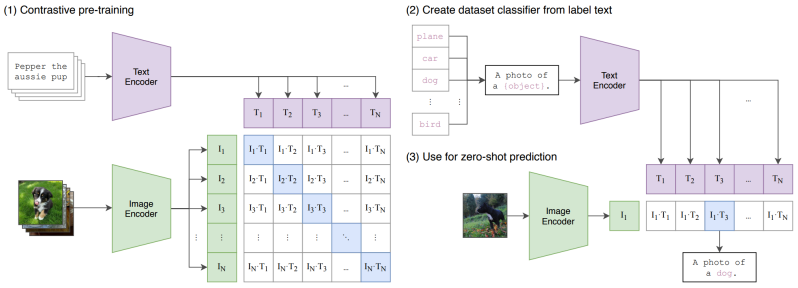


Figure 27: (1) Contrastive pre-training stage of a CLIP model (training stage). (2) and (3): after the model has been pre-trained, it is used as a zero-shot image classifier [63].

them with weight that indicates how much we want the network to pay attention to the conditioning signal.

The formulation for classifier-free guidance is given by:

$$\nabla \log p(x_t|y) = \underbrace{\gamma \nabla \log p(x_t|y)}_{\text{conditional score}} + \underbrace{(1-\gamma)\nabla \log p(x_t)}_{\text{unconditional score}}$$

In contrast to classifier guidance, classifier-free guidance streamlines the training process and lowers computational costs by utilizing a single model instead of training two separate models.

Listing 2: Classifier-free guidance (CFG) in Stable Diffusion.

```
if do_cfg:
    output_cond, output_uncond = model_output.chunk(2)
    model_output = cfg_scale * (output_cond - output_uncond) + output_uncond
```

In listing 2 the code snippet is taken from a re-implementation of Stable Diffusion, but the official implementation of Stable Diffusion is very similar.

8.6 Contrastive Language Image Pre-training (CLIP)

In Stable Diffusion, the authors used CLIP as the text encoder for the conditional text prompts.

CLIP (Contrastive Language Image Pre-training) [63] is a model developed by OpenAI that learns visual concepts from text supervision. The model builds **associations between images and text prompts**. Often this association is called 'text-image alignment', or 'image-text alignment'. It models how well a generated image corresponds to its associated text description.

The **CLIPTokenizer**, which is part of the CLIP model, converts text prompts to a sequence of tokens which are then used in the latent space of the Stable Diffusion model.

In figure 27, $I_1, ..., I_N$ are the images, and $T_1, ..., T_N$ are the text prompts. The output is a **matrix of similarity scores** between the images and the text descriptions. Ideally in the matrix diagonal we get high similarity score (1) indicating high text-image alignment, and all other entries ideally should have low similarity score (0), indicating mismatch in text-image alignment.

The CLIP network is compromised of:

- **Image encoder**: converts images to image embeddings. It is typically a vision transformer (ViT) [20] (appendix F) or a ResNet [27] model.
- **Text encoder**: converts text descriptions to text embeddings. It is typically implemented as a transformer, or less common as a continuous bag of words [51] (known as Word2Vec model by Google, 2013 [51]).
- **Training objective**: the CLIP model is trained using a contrastive objective (commonly referred to as CLIP objective), where the goal is to minimize the cosine distance in the main diagonal and maximize the distance for non-matching image-text pairs (off diagonal).

8.7 DDIM Sampler

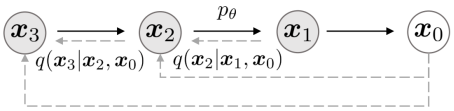


Figure 28: Non-Markovian inference in DDIM. Each step depends on both the previous step and the initial state x_0 [84].

Denoising Diffusion Implicit Models (DDIM), introduced in [84], provide an **efficient sampling method** for diffusion models, offering improvements over the traditional Denoising Diffusion Probabilistic Models (DDPM) [31]. Unlike DDPM sampler, which uses a fixed noise schedule and stochastic sampling, DDIM introduces a flexible noise schedule and a **deterministic sampling process**, enabling faster inference (10x to 50x speedup compared to DDPM [84]).

Key points & main contributions of the DDIM paper:

- **Non-Markovian process**: As illustrated in figure 28, DDIM's reverse process depends on both the previous step and the initial state x_0, forming a deterministic trajectory through the latent space.
- **Implicit probabilistic model**: DDIM is an implicit probabilistic model, meaning that it doesn't directly model the joint distribution of the data but rather models the conditional distribution of the data given the noise.
- **Deterministic sampling**: Noise is removed directly in a controlled manner, skipping unnecessary diffusion steps and avoiding stochasticity.
- **Training objective**: DDIM uses the same training objective as DDPM and no modifications are needed.
- **Sampling process**: Sampling in DDIM involves sampling from the prior distribution and then iteratively sampling from the conditional distributions. This process is faster than traditional diffusion models because it doesn't require simulating the entire Markov chain.

DDIM introduces a generalized forward process:

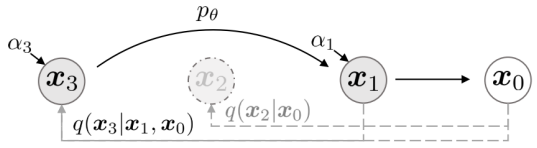


Figure 29: DDIM sampling [84]. In the figure, x_3 depends only on x_0 and x_1 (and not x_2). This process can be generalized to all subsets of steps.

		CIFAR10 (32 × 32)					CelebA (64 × 64)				
	S	10	20	50	100	1000	10	20	50	100	1000
η	0.0	**13.36**	**6.84**	**4.67**	**4.16**	4.04	**17.33**	**13.73**	**9.17**	**6.53**	3.51
	0.2	14.04	7.11	4.77	4.25	4.09	17.66	14.11	9.51	6.79	3.64
	0.5	16.66	8.35	5.25	4.46	4.29	19.86	16.06	11.01	8.09	4.28
	1.0	41.07	18.36	8.01	5.78	4.73	33.12	26.03	18.48	13.93	5.98
	$\hat{\sigma}$	367.43	133.37	32.72	9.99	**3.17**	299.71	183.83	71.71	45.20	**3.26**

Figure 30: DDIM gives almost the same sample quality compared to DDPM sampler, and requires less compute since we skip some diffusion steps. The stochasticity of the model is controlled by $\eta \in [0, 1]$ [84] (the equations where η is used are not shown in this work for simplicity).

$$q_\sigma(x_{t-1}|x_t, x_0) = \mathcal{N}\left(\sqrt{\alpha_{t-1}}x_0 + \sqrt{1 - \alpha_{t-1} - \sigma_t^2} \cdot \frac{x_t - \sqrt{\alpha_t}x_0}{\sqrt{1 - \alpha_t}}, \sigma_t^2 I\right)$$

where $\sigma \in \mathbb{R}_{\geq 0}^T$ (a real vector of T dimension, each element greater or equal to 0) controls the stochasticity of the forward process. When $\sigma \to 0$, the process becomes fully deterministic, as x_{t-1} is uniquely determined by x_t and x_0.

In figure 29 we see the DDIM sampling process skips a single diffusion step. We can skip some diffusion steps and this process can be generalized to all the diffusion steps.

In figure 30 DDIM gives almost the same sample quality in FID metric (equation 1, lower is better) compared to DDPM and require much less compute. The experiment was done on CIFAR10 and CelebA datasets, with 10,20,50,100,1000 steps. When $\eta = 0$ (blue) the model is deterministic (DDIM process), and when $\eta = 1$ (orange) the model is stochastic (DDPM process).

8.8 Training

Stable Diffusion is trained in CFG manner (section 8.5).

The authors made simplified loss objective which predicts the noise removal process at each step:

$$L_{\text{DM}} = \mathbb{E}_{x,\epsilon\sim\mathcal{N}(0,1),t}\left[\|\epsilon - \epsilon_\theta(x_t, t)\|_2^2\right]$$

This loss function is similar to the loss function of DDPM (equation 14).

8.9 Implementation of τ_θ transformer for conditional LDMs

The researchers provided high level overview of the implementation of the conditional encoder for text τ_θ, consisted of N transformer blocks (appendix E.2 of [71]):

$$
\begin{aligned}
&\zeta \leftarrow \text{TokEmb}(y) + \text{PosEmb}(y) \\
&\text{for } i = 1, \dots, N: \\
&\qquad \zeta_1 \leftarrow \text{LayerNorm}(\zeta) \\
&\qquad \zeta_2 \leftarrow \text{MultiHeadSelfAttention}(\zeta_1) + \zeta \\
&\qquad \zeta_3 \leftarrow \text{LayerNorm}(\zeta_2) \\
&\qquad \zeta \leftarrow \text{MLP}(\zeta_3) + \zeta_2 \\
&\zeta \leftarrow \text{LayerNorm}(\zeta)
\end{aligned}
$$

where:

- $\zeta := \tau_\theta(y)$ is the unmasked transformer output (the transformer processes the tokenized version of y), which is then used in the cross-attention mechanism of the stable diffusion model.
- TokEmb is the token embeddings.
- PosEmb is the positional embeddings.
- LayerNorm is the layer normalization (appendix C.3).
- MultiHeadSelfAttention is the multi-head self-attention mechanism (appendix D).
- MLP is the multi-layer perceptron (appendix C) block.

8.10 Details on Autoencoders Models

In the stable diffusion paper, the researchers trained the pixel-space encoder ε and the decoder $\mathcal{D}$ in adversarial manner (adversarial loss [21]). A patch-based discriminator D_ψ is optimized to differentiate the original image from reconstructed image $\mathcal{D}(\varepsilon(x))$ which helps guide the autoencoder.

In addition, the researchers used two methods for regularizing the latent space:

- **KL-divergence regularization**: similar to VAEs where the latent space is a distribution, the KL regularization objective pushes the latent space to be close to a standard normal distribution.
- **VQ regularization**: similar to VQ-GAN and VQ-VAE (sections 6 and 4), the vector quantization regularization forces the latent space to be discrete, which can help the model to learn better by using a codebook $\mathcal{Z}$.

They factorized the KL term by a factor of $\sim 10^{-6}$, and in VQ regularization they used a big codebook.

The full loss objective to train the autoencoder model $(\varepsilon, \mathcal{D})$ is given by:

$$
L_{\text{Autoencoder}} = \min_{\varepsilon, \mathcal{D}} \max_{\psi} \left(L_{\text{rec}}(x, \mathcal{D}(\varepsilon(x))) - L_{\text{adv}}(\mathcal{D}\varepsilon(x)) + \log \mathcal{D}_\psi(x) + L_{\text{reg}}(x; \varepsilon, \mathcal{D}) \right)
$$

CelebA-HQ 256 × 256				FFHQ 256 × 256			
Method	FID ↓	Prec. ↑	Recall ↑	**Method**	FID ↓	Prec. ↑	Recall ↑
DC-VAE [63]	15.8	-	-	ImageBART [21]	9.57	-	-
VQGAN+T. [23] (k=400)	10.2	-	-	U-Net GAN (+aug) [77]	10.9 (7.6)	-	-
PGGAN [39]	8.0	-	-	UDM [43]	5.54	-	-
LSGM [93]	7.22	-	-	StyleGAN [41]	4.16	0.71	0.46
UDM [43]	7.16	-	-	ProjectedGAN [76]	**3.08**	0.65	0.46
LDM-4 (ours, 500-s†)	**5.11**	0.72	0.49	*LDM-4* (ours, 200-s)	4.98	**0.73**	**0.50**

LSUN-Churches 256 × 256				LSUN-Bedrooms 256 × 256			
Method	FID ↓	Prec. ↑	Recall ↑	**Method**	FID ↓	Prec. ↑	Recall ↑
DDPM [30]	7.89	-	-	ImageBART [21]	5.51	-	-
ImageBART [21]	7.32	-	-	DDPM [30]	4.9	-	-
PGGAN [39]	6.42	-	-	UDM [43]	4.57	-	-
StyleGAN [41]	4.21	-	-	StyleGAN [41]	2.35	0.59	0.48
StyleGAN2 [42]	3.86	-	-	ADM [15]	1.90	**0.66**	**0.51**
ProjectedGAN [76]	**1.59**	0.61	0.44	ProjectedGAN [76]	**1.52**	0.61	0.34
*LDM-8** (ours, 200-s)	4.02	**0.64**	**0.52**	*LDM-4* (ours, 200-s)	2.95	**0.66**	0.48

Figure 31: Unconditional image synthesis evaluation between LDM (Stable Diffusion) and other models across 4 datasets [71]. † refers to the DDIM sampler steps (500 top-left, 200 top-right, 200 bottom-left, 200 bottom-right).

Text-Conditional Image Synthesis				
Method	FID ↓	IS↑	N_{params}	
CogView† [17]	27.10	18.20	4B	self-ranking, rejection rate 0.017
LAFITE† [109]	26.94	26.02	75M	
GLIDE* [59]	12.24	-	6B	277 DDIM steps, c.f.g. [32] $s = 3$
Make-A-Scene* [26]	**11.84**	-	4B	c.f.g for AR models [98] $s = 5$
LDM-KL-8	23.31	$20.03_{\pm 0.33}$	1.45B	250 DDIM steps
*LDM-KL-8-G**	12.63	$\mathbf{30.29}_{\pm \mathbf{0.42}}$	1.45B	250 DDIM steps, c.f.g. [32] $s = 1.5$

Figure 32: Evaluation of text-conditioned image synthesis on MS-COCO dataset. LDM with 250-DDIM steps is on par with the most recent diffusion and autoregressive methods, **while using significantly fewer parameters** (1.45 billion) [71].

8.11 Experiments

The researchers conducted several experiments with different LDM downsampling factors, and compared the results with other state-of-the-art models (DALL-E, VQ-GAN, StyleGAN, ProjectedGAN, CogView, GLIDE, and others).

In figure 31 we clearly see that LDM outperforms most of the state-of-the-art models, across multiple metrics and datasets.

In figure 32 we see comparison between different methods on the text conditional image synthesis task. While we desire lower FID and higher IS metrics, we also need to look at number of parameters: LDM with 250-DDIM steps is on par with the most recent diffusion and autoregressive methods, while using significantly fewer parameters (1.45 billion).

For text-to-image tasks, the researchers trained a 1.45B parameters model conditioned on language prompts on LAION-400M dataset. The model uses **BERT-Tokenizer** ¶ [18] and implement

¶It is important to note that the researchers used the BERT text tokenizer in the paper, however, in the official released implementation of Stable Diffusion they used CLIP tokenizer because in the Imagen paper [75], the researchers found out that using larger language models had more impact on generated image quality than larger

Figure 33: Samples of Stable Diffusion conditioned on semantic layouts [71].

τ_θ (the domain-specific conditional encoder) as a transformer.

For semantic layouts, the model is trained to synthesis images based on semantic layouts from the OpenImages dataset and fine-tuned on COCO dataset (figure 33).

In figure 41, which is from Imagen paper [75], we see that using larger text encoder has more impact on image quality than larger image generation components (like the U-Net). This is why in Stable Diffusion, the implementation uses CLIP tokenizer instead of BERT tokenizer (which was originally used in the paper).

9 Imagen

Imagen [75] is a T2I diffusion model that builds on the power of large transformer language models [97] (LLMs) to generate high-fidelity images. The combination of diffusion and LLMs have shown remarkable outputs.

The paper has 5 main key takeaways and observations:

1. **Effectiveness of large frozen text encoders**[‖]: one of the main observation in the paper that a large frozen language model trained only on text data have a significant impact on the fidelity of generated images compared to increasing the parameters of the diffusion image model. Scaling the language model is easy, since unlabeled text data is abundant and available on the internet.

2. **Dynamic thresholding**: is a new sampling technique (section 9.3) that improves image fidelity and text-image alignment, which improves upon static thresholding.

3. **Effective U-Net**: a new U-Net architecture that is simpler and more memory efficient.

4. **COCO FID score of 7.27**: imagen achieved a new state-of-the-art COCO FID score of 7.27, which outperforms all other previous works.

image generation components. This finding is shown in figure 41.

[‖]Frozen text encoder means that the parameters of the text encoder are frozen; they won't change in training. It is done usually when we want to use a pre-trained model (text encoder) while training our own model. It is also faster to train since gradient calculation will skip this frozen model.

5. **DrawBench**: a new human evaluation benchmark for T2I task. Imagen outperforms all other works, including DALL-E [67], VQ-GAN+CLIP [17], LDM [71], GLIDE [54], and DALL-E 2 [66].

9.1 Text-to-Text Transfer Transformer (T5)

Text-**t**o-**T**ext **T**ransfer **T**ransformer (T5) [64] by Google Research is a language model that treats tasks as a text-to-text (T2T) problems. For example, we could prompt the model:

- **Summarization**: "Translate the following text to a summary: ..."
- **Translation**: "Translate the following text from English to French: ..."
- **Text classification**: "Classify the following text into one of the following categories: ..."
- **Question answering**: "Answer the following question: ..."

as well as other T2T tasks. In short, this knowledge can be viewed as developing a 'general-purpose' model that can understand text, instead of explicitly training the model to complete the specific downstream task.

The T5 model is open-source and was trained on large corpora of textual data. The base version of the model (T5-base) consists of 220 million parameters, while the largest version of the model (T5-XXL) consists of **11 billion parameters**. In the context of Imagen, **the Imagen model uses a frozen version of T5-XXL model** to encode conditional text prompts.

Pre-training: Unsupervised learning is appealing because unlabeled text data is abundant and available on the Internet. For example, the Common Crawl project [16] is a non-profit organization that crawls the internet and provides free access to it is achieved datasets to the public. A lot of research has been done on LLM models on large scale datasets, and the consensus is that **the larger the dataset, the better the model performs** [62] [40] [28]. The T5 models were trained on the "Colossal Clean Crawled Corpus" (C4) dataset, which consists of 750 GB of English text data scraped from the web.

The training objective of T5 model is called **span corruption**, which is stronger version of **masked language modeling**. Given a sentence, some words and some contiguous words are masked (in masked language modeling, only single words are masked), and the model should predict those words. For example: "Thank you for inviting me to your party last week", where the masked words are "for inviting" and "last". And the model should predict those words in the following sentence: "Thank you [MASKED] me to your party [MASKED] week". The model should learn to reconstruct the missing tokens.

In figure 34 <M> denotes shared mask token (the same mask token is used to represent all masked positions in the input). <X>, <Y>, and <Z> denote sentinel tokens which have unique token IDs; they mark specific masked positions that the model should reconstruct.

9.2 Pre-trained text encoders

In Imagen [75] the researchers explored some of the biggest and most advanced text encoders: **T5-XXL** [64], **GPT** [12] [61] [60], and **BERT** [18]. These LLMs were trained exclusively on text datasets, which are substantially larger compared to image-text pair datasets (as used in models like **CLIP**).

Objective	Inputs	Targets
Prefix language modeling	Thank you for inviting	me to your party last week .
BERT-style Devlin et al. (2018)	Thank you <M> <M> me to your party apple week .	*(original text)*
Deshuffling	party me for your to . last fun you inviting week Thank	*(original text)*
MASS-style Song et al. (2019)	Thank you <M> <M> me to your party <M> week .	*(original text)*
I.i.d. noise, replace spans	Thank you <X> me to your party <Y> week .	<X> for inviting <Y> last <Z>
I.i.d. noise, drop tokens	Thank you me to your party week .	for inviting last
Random spans	Thank you <X> to <Y> week .	<X> for inviting me <Y> your party last <Z>

Figure 34: Mask modeling in T5 [64].

Freezing these models ** provides significant advantage over training them: less memory and compute costs.

9.3 Diffusion guidance weight

As described before (section 8.5), there are two methods to increase sample quality with the tradeoff of diversity:

- **Classifier guidance** uses a separate, pre-trained classifier model to guide the image generation process in diffusion models by adjusting the noise based on how closely the generated image matches a desired condition.
- **Classifier-free guidance** (CFG) removes the need for a separate classifier by training the diffusion model itself to optionally condition on the label or text input.

More formally, in CFG, sampling is performed weighting the conditional and unconditional signals:

$$\underbrace{\tilde{\epsilon}_\theta(z_t, c)}_{\text{adj noise prediction}} = \underbrace{w\epsilon_\theta(z_t, c)}_{\text{conditional score}} + \underbrace{(1-w)\epsilon_\theta(z_t)}_{\text{unconditional score}} \tag{16}$$

where ϵ_θ is the noise prediction with the learned parameters θ, w is the guidance weight, c is the condition, and z_t is the latent variable at timestep t.

Imagen depends on CFG for effective text conditioning. They found that increasing the guidance weight improves image-text alignment but reduces image fidelity. It is caused by **train-test mismatch**: looking at equation 16 if we set w to 1, then we disable CFG, and then the model won't be trained on unconditional samples (only on conditional signals). This causes the model to generate outputs that exceed the noise prediction in the normalized range of [-1, 1]. And when iteratively applying the model to its own outputs at each step, errors caused by this mismatch accumulate, which causes unnatural artifacts in output images. When we set $w > 1$ then it improves the model's image-text alignment, but damages image fidelity.

For this reason they investigated static thresholding and dynamic thresholding:

**When we freeze models we generally mean that some (or all) of the parameters of the model are not changed during training. How? When a layer is frozen during training, no gradient updates will occur for this layer. Gradients will still flow from frozen layer to non-frozen layer, it doesn't skip the backpropagation. It just passes the gradients from the next layer to the previous layer.

```
def sample():
  for t in reversed(range(T)):
    # Forward pass to get x0_t from z_t.
    x0_t = nn(z_t, t)

    # Static thresholding.
    x0_t = jnp.clip(x0_t, -1.0, 1.0)



    # Sampler step.
    z_tm1 = sampler_step(x0_t, z_t, t)
    z_t = z_tm1
  return x0_t
```

```
def sample(p: float):
  for t in reversed(range(T)):
    # Forward pass to get x0_t from z_t.
    x0_t = nn(z_t, t)

    # Dynamic thresholding (ours).
    s = jnp.percentile(
        jnp.abs(x0_t), p,
        axis=tuple(range(1, x0_t.ndim)))
    s = jnp.max(s, 1.0)
    x0_t = jnp.clip(x0_t, -s, s) / s

    # Sampler step.
    z_tm1 = sampler_step(x0_t, z_t, t)
    z_t = z_tm1
  return x0_t
```

Figure 35: Static (left) and dynamic (right) thresholding code implementation [75].

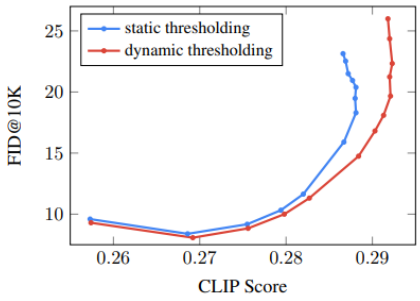


Figure 36: Static vs dynamic thresholding [75].

Static thresholding applies the clipping operation to force the x-prediction output to fit to the normalized range of [-1, 1]. However, static thresholding still results in over-saturated and less detailed images as the guidance weight approaches 1.

Dynamic thresholding: instead of clipping the x-prediction to the normalized range, it sets a dynamic threshold s based on the distribution of absolute pixel values in the current x-prediction, allowing the threshold to adapt to the specific output of the model at that moment. In other words, if the pixel values are saturated (close to the [-1, 1] range), dynamic thresholding pushes them inwards by thresholding in the range [$-s$, s] and then dividing by s.

The implementation of static and dynamic thresholding is shown in figure 35. The impact of dynamic thresholding is shown in figure 36 where it's shown that dynamic thresholding produces samples with higher image-text alignment (CLIP) and fidelity (FID) compared to static thresholding.

9.4 Super-resolution via Repeated Refinement (SR3)

In a 2021 paper [74], the Google Research team introduced a new super-resolution model based on diffusion process. **S**uper-resolution via **R**epeated **R**efinment (SR3) model upsamples in iterative manner, similar to reverse diffusion. It upsamples images from 64×64 to 256×256 and finally to 1024×1024.

SR3 achieves close to a 50% fool rate (47%) [††] on $16 \times 16 \rightarrow 128 \times 128$ faces, outperforming the

[††] A 50% fool rate means humans can't distinguish between a generated face image and an image of a real face.

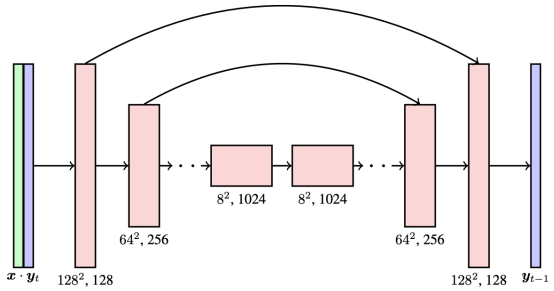


Figure 37: SR3 $16 \times 16 \rightarrow 128 \times 128$ upsampling where x is the input image, upscaled to the target resolution using bicubic interpolation, and then it is concatenated with the noise y [74].

previous state-of-the-art GAN models (FSRGAN and PULSE).

Iterative refinement: Given low-resolution image x_{low} the model applies **bicubic interpolation** [‡‡] to upscale the image to the target resolution to get high-resolution image x and then concatenates x with pure Gaussian noise $y_t \sim \mathcal{N}(0, I)$ channel-wise (see figure 37). Then the U-Net iteratively refines the image (x, y_t) by denoising. The output of a single refinement step (from the U-Net) is y_{t-1}. Then in the next refinement step, x is concatenated with y_{t-1} and the process is repeated until we reach y_0. The final output is a high-resolution image y_0. Note that both x (after the upsampling), y_i are all on same resolution / dimension (see figure 37).

The architecture of SR3 modified the diffusion U-Net: they replaced the original DDPM residual blocks with residual blocks from BigGAN [10], rescaled skip connections by $\frac{1}{\sqrt{2}}$, increased the number of residual blocks, and increased the number of the channel multipliers [§§] at different resolutions.

9.5 Cascaded diffusion models (CDMs)

Cascaded diffusion models, introduced in a 2022 paper [34] by Google Research, builds upon the SR3 paper [74] and introduces a new method for super-resolution (SR) called Cascaded Diffusion Models (CDMs). CDMs use a base diffusion model that output image at low resolution, followed by a sequence (pipeline) of SR3 SR diffusion models that progressively upsamples until we reach the target resolution.

In the CDM paper [34], the model outperformed VQ-VAE 2 [68] and BigGAN-deep [10] in SR task in FID metric on ImageNet dataset.

A big strength of CDMs is the ability to train and fine tune each model individually.

Conditioning augmentation is the main and critical part of the paper [34], which allows generating images with higher FID, compared to without conditioning augmentation. Conditioning augmentation involves using data augmentation techniques on the low-resolution input image for each SR model in the cascaded pipeline. Augmentations such as Gaussian noise and Gaussian blur

[‡‡]Bicubic interpolation is a method for resizing images that uses the closest 4x4 pixels grid to estimate new values, resulting in smoother transitions (compared to nearest-neighbor or bilinear interpolation) and fewer visual artifacts.

[§§]Channel multipliers in a U-Net are the scaling factors used to adjust the number of feature channels at different U-Net layers. In other words, they increased the depth of the features at the cost of decreasing resolution at the convolution layers (down convolution, up convolution layers)

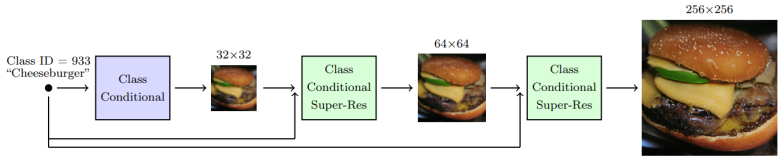


Figure 38: CDM upsamples low-resolution image in the pipeline, conditioned on labels [34].

are applied to the image which helps prevent each SR model from overfitting to the low-resolution input.

In figure 38 we see CDM upsamples the low-resolution image in multiple steps with class conditional super-resolution; the condition is class ID = "Cheeseburger" which guides the generation pipeline to upsample correctly.

9.6 Architecture

An overview of Imagen architecture is shown in figure 40.

The researchers conducted experiments with frozen LLMs such as BERT [18], T5 [64], and CLIP [63] and found that humans prefer T5-XXL over CLIP. T5-XXL (T5-Extra Extra Large) text encoder maps text prompts to embeddings. The T5-XXL model has 11 billion parameters, and it's the largest version of the T5 model. All the diffusion models are conditioned on the same text embeddings.

Base model: the base model is a 64×64 T2I diffusion model. The network is conditioned on text embeddings (via cross-attention and layer normalization layers), as well as with diffusion timestep embeddings, similar to the class embedding conditioning in the CDM paper [34].

SR models: they used modified U-Net based on the improved DDPM paper [55] by OpenAI, By improving the U-Net they achieved 2-3x faster inference and convergence speed; they call this variant "**Efficient U-Net**". For the $256 \times 256 \rightarrow 1024 \times 1024$ SR model, they removed self-attention layers but keep the cross-attention layers.

Efficient U-Net: is a new U-Net architectural variant for SR models. It is more memory efficient and **2-3x faster in training and inference time**. There are several modifications to the U-Net architecture:

1. **More residual blocks**: adding more residual blocks for lower resolutions, since lower-resolution images typically have more channels. They used 8 residual blocks at lower-resolution compared to typical 2-3 residual blocks used in a standard U-Net.

2. **Scaling the skip connections** by $\frac{1}{\sqrt{2}}$, similar to SR3 [74], significantly improves convergence speed.

3. **Reversing order of downsampling and upsampling blocks**: In a standard U-Net downsampling block, downsampling occurs after the convolution layers, and in the upsampling block, upsampling is done before the convolutions. They reverse this order for both blocks which significantly improves the forward pass without performance degradation.

The efficient U-Net architecture for $64 \times 64 \rightarrow 256 \times 256$ is shown in figure 39.

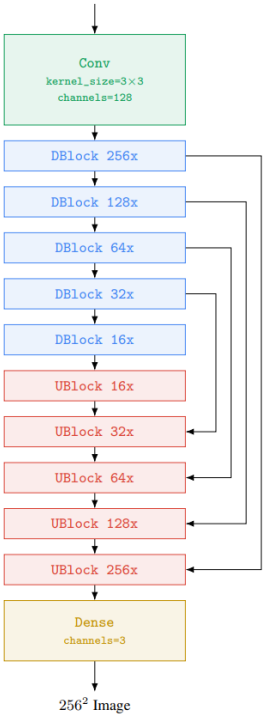


Figure 39: Efficient U-Net architecture for $64 \times 64 \to 256 \times 256$ SR model, proposed in Imagen [75].

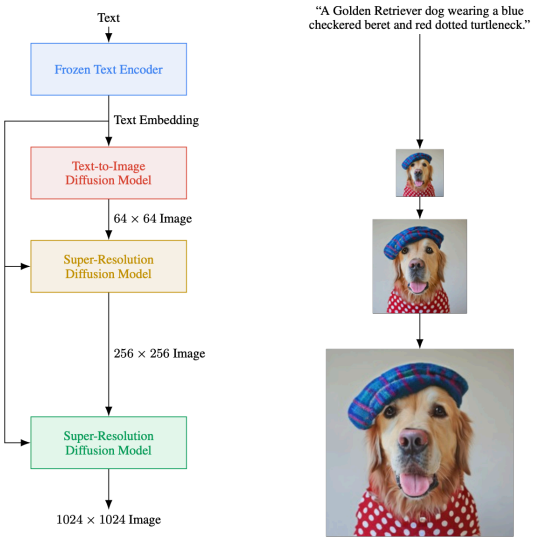


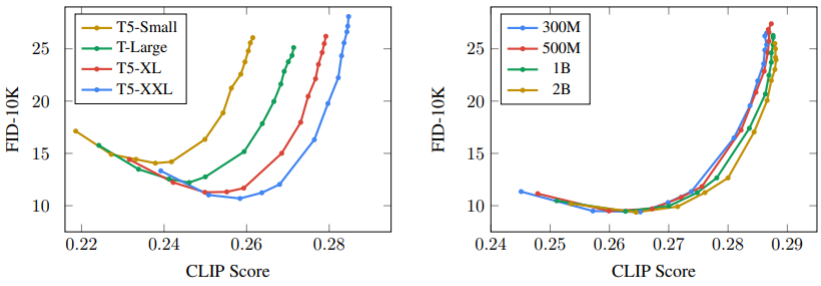
Figure 40: Overview of Imagen architecture [75].

Similar to LDMs, Imagen also uses CFG [32] (section 8.5). Imagen depends critically on CFG for effective text conditioning.

Noise conditioning: Imagen corrupts the 64×64 image with Gaussian noise. The amount of noise is random at training but arbitrary at inference time. They control the amount of corruption with *aug_level*, and the SR model is conditioned on the augmentation level.

Number of parameters: the base T2I diffusion 64×64 model has 2 billion parameters. The $64 \times 64 \rightarrow 256 \times 256$ SR model has 600 million parameters, and the $256 \times 256 \rightarrow 1024 \times 1024$ SR model has 400 million parameters. The size of the T5-XXL text encoder is 4.6 billion parameters.

Scaling text encoder size is more important than U-Net size: the researchers found that scaling the text encoder has significantly more impact than increasing U-Net size: see figure 41.



(a) Impact of encoder size. (b) Impact of U-Net size.

Figure 41: Scaling the encoder size is more impactful than scaling the U-Net size [75].

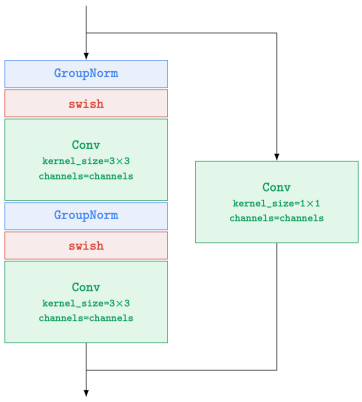
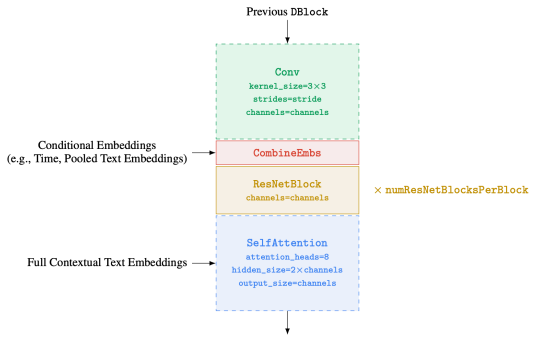
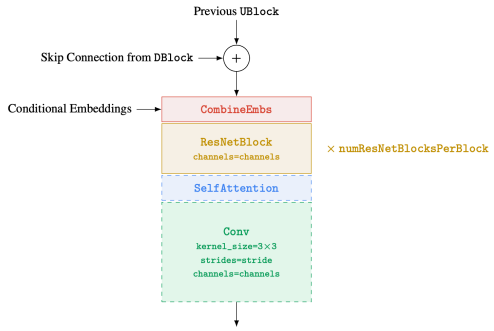


Figure 42: Imagen efficient U-Net `ResNetBlock` architecture [75].

Figure 43: Efficient `UBlock` and `DBlock` architectures proposed in Imagen paper [75].



(a) Imagen efficient U-Net `DBlock` architecture [75].



(b) Imagen efficient U-Net `UBlock` architecture [75].

In figure 42 we can see the `ResNetBlock` which is in use by both the `DBlock` and the `UBlock`. The only input of the `ResNetBlock` is the number of channels. The activation function is Swish (equation 18).

And in figure 43 we see the architecture of `UBlock` and `DBlock`.

9.7 DrawBench

The COCO dataset [48] is a standard benchmark for evaluating T2I models. The standard performance metrics used are FID [29] which measure image fidelity (but is not fully aligned with perceptual quality [57]), and CLIP score [63] which measures image-text alignment (but is bad at counting objects in an image).

Due to these limitations, the researchers created new benchmark called **DrawBench** that uses human evaluation to assess image quality by asking "Which image is more photorealistic (looks more real)?" and text-image alignment by asking "Does the caption accurately describe the above

Category	Description	Examples
Colors	Ability to generate objects with specified colors.	"A blue colored dog." "A black apple and a green backpack."
Counting	Ability to generate specified number of objects.	"Three cats and one dog sitting on the grass." "Five cars on the street."
Conflicting	Ability to generate conflicting interactions b/w objects.	"A horse riding an astronaut." "A panda making latte art."
DALL-E [53]	Subset of challenging prompts from [53].	"A triangular purple flower pot." "A cross-section view of a brain."
Description	Ability to understand complex and long text prompts describing objects.	"A small vessel propelled on water by oars, sails, or an engine." "A mechanical or electrical device for measuring time."
Marcus et al. [38]	Set of challenging prompts from [38].	"A pear cut into seven pieces arranged in a ring." "Paying for a quarter-sized pizza with a pizza-sized quarter."
Misspellings	Ability to understand misspelled prompts.	"Rbefraigerator." "Tcennis rpacket."
Positional	Ability to generate objects with specified spatial positioning.	"A car on the left of a bus." "A stop sign on the right of a refrigerator."
Rare Words	Ability to understand rare words[3].	"Artophagous." "Octothorpe."
Reddit	Set of challenging prompts from DALLE-2 Reddit[4].	"A yellow and black bus cruising through the rainforest." "A medieval painting of the wifi not working."
Text	Ability to generate quoted text.	"A storefront with 'Deep Learning' written on it." "A sign that says 'Text to Image'."

Figure 44: Descriptions and examples of the 11 categories in DrawBench [75].

Model	FID-30K	Zero-shot FID-30K
AttnGAN [76]	35.49	
DM-GAN [83]	32.64	
DF-GAN [69]	21.42	
DM-GAN + CL [78]	20.79	
XMC-GAN [81]	9.33	
LAFITE [82]	8.12	
Make-A-Scene [22]	7.55	
DALL-E [53]		17.89
LAFITE [82]		26.94
GLIDE [41]		12.24
DALL-E 2 [54]		10.39
Imagen (Our Work)		**7.27**

(a) Although Imagen was not explicitly trained on MS-COCO dataset, it outperforms all other models on COCO FID score, achieving 7.27 FID.

Model	Photorealism ↑	Alignment ↑
Original		
Original	50.0%	91.9 ± 0.42
Imagen	$39.5 \pm 0.75\%$	91.4 ± 0.44
No people		
Original	50.0%	92.2 ± 0.54
Imagen	$43.9 \pm 1.01\%$	92.1 ± 0.55

(b) Human evaluation on 256x256 COCO. Imagen splits to two categories: no filters, and human filters. Imagen struggles a little with photorealistic people.

Figure 45: Results of Imagen on COCO dataset [75].

image?". For both cases they used 200 randomly chosen image-caption pairs from COCO dataset.

DrawBench has 11 categories of prompts which test different capabilities of models such as ability to render different colors, number of objects, spatial relations, text in the scene, and more. The prompts include long captions, rare words, and misspelled prompts. See figure 44 for examples of the categories and examples.

9.8 Results

In figure 45a we can see that although Imagen was not trained on the MS-COCO dataset, it still outperforms state-of-the-art models such as DALL-E 2 [66] and achieves the best MS-COCO FID score of 7.27 in zero-shot setting.

In figure 45b we can see that Imagen achieves a respectable 43.9% fool rate on 256×256 COCO human faces, indicating limited ability of Imagen to generate photorealistic people.

And in figure 46 we can see that Imagen outperforms all other state-of-the-art models in terms

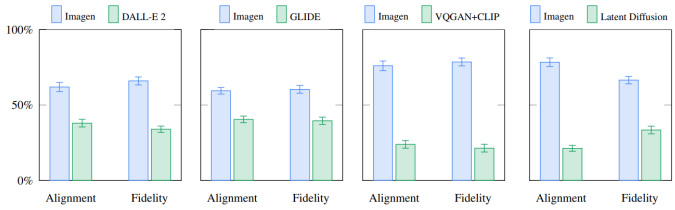


Figure 46: Imagen beats all other models (DALL-E 2 [66], GLIDE [54], VQ-GAN+CLIP [17] and Latent Diffusion (Stable Diffusion) [71] (section 8)) in terms of image-text alignment and image fidelity [75].

of image fidelity and text-image alignment.

10 Video Synthesis

Video synthesis is a complex task. One can think of video generation as a sequence of image generation tasks. Formally, a video is a sequence of images (or frames) that are shown in fast fashion, usually 24 frames per second (at the minimum, to get smooth video). Therefor to create a smooth video of 5 seconds, you'll need 120 frames or images at the minimum. Video data is complex; we have the addition of the **time dimension**, which is not present in image generation tasks. The video should be **coherent** in time, meaning that the frames should be related to each other and should follow a logical sequence. Objects should not appear out of nowhere, there should be **smooth transition of motion** and correct **spatial relationships** between objects. We also have to deal with limited hardware resources, since video generation is extremely computationally expensive.

Video synthesis can be based on four main methods:

1. **Diffusion**: diffusion models for video generation extend the iterative refinement process used in image-based diffusion models, such as LDMs, to handle temporal dynamics required for videos. The key idea is to generate not just a sequence of independent images, but a continuous video where both the spatial content of each frame and the temporal coherence between frames are learned and generated simultaneously.

2. **Spatial autoregressive**: spatial autoregressive models for video generation, as described in [25], synthesize video by sequentially generating content in a patch-based fashion, where each patch in a frame is conditioned on previously generated patches. This method builds video frame-by-frame, ensuring spatial and temporal coherence.

3. **GAN**: similar to diffusion models, GANs can also be leveraged for video synthesis. However, it's used seldom because of unstable training, partly due to the adversarial loss objective.

4. **Mask modeling**: mask modeling in video generation uses the technique of selectively masking parts of video frames. The model is tasked with predicting and reconstructing the missing parts, forcing it to better understand the underlying structure and motion in the video.

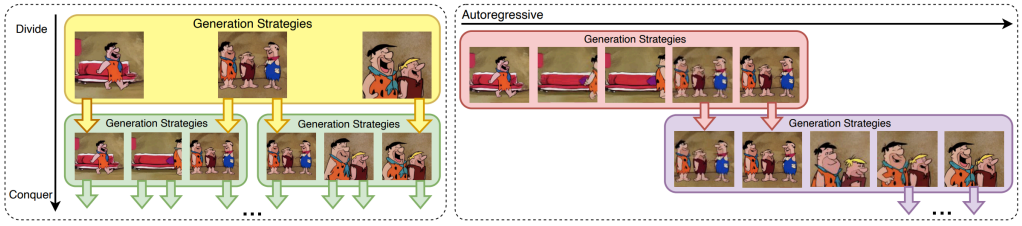


Figure 47: Overview of video synthesis paradigms [47]. Divide and conquer is split into hierarchical architecture for keyframe generation and frame filling. Whereas temporal autoregressive paradigm the next frame prediction depends on the previous frame/frames [47].

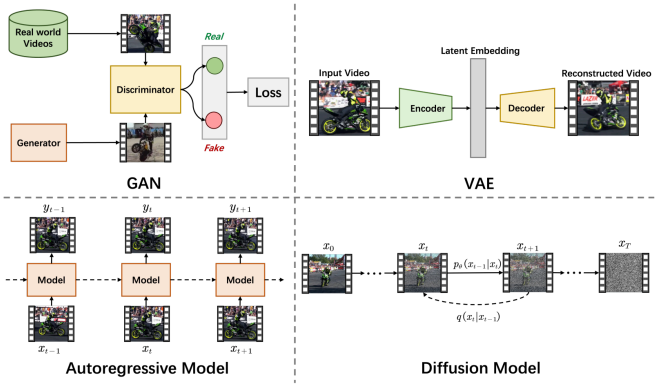


Figure 48: Overview of video generation technologies [112].

In the realm of video synthesis there are two paradigms: **divide and conquer** and **temporal autoregressive**. An overview of the paradigms is shown in figure 47.

In figure 48 we can see different video generation techniques, some we already covered: GAN, VAE, diffusion and autoregressive. The autoregressive model we did not yet covered; it is covered in the next sections.

Temporal autoregressive

Temporal autoregressive paradigm is an iterative approach where each frame is generated based on the previous one (or previous multiple frames). The model is trained on sequences of video data, and the output from one timestep serves as the input for the next (the next frame is conditioned on the previous frame, which in theory ensures that the video is temporally coherent).

A significant amount of research has been conducted on temporal autoregressive models, leading to various improvement strategies and enhancements for this framework. Some of the most notable include:

- **Latent space compression**: this method focuses on compressing videos to latent space to optimize computational needs. For example in [109] and [26] they explored condensing video data into 3D latent. Compression aims to preserve essential features across dimensions to improve computational efficiency.
- **Incorporating temporal layers** to refine individual video clips. Advancements were made, such as integrating **temporal layers** such as attention and convolutional layers into diffusion models, like in VideoLDM [8] and [26]. These temporal layers help the model better understand the temporal dynamics of video data.
- **Dual-phase training & reuse strategies**: dual-training is a training strategy where the model is first trained unconditionally, and then learn to conditionally generate video like in Projected Latent Video Diffusion Model (PVDM) [108]. Given the abundance of unlabeled video data on the web, it's easy to see why this process is attractive. Another method to boost the model's ability to replicate long video sequences is a **reuse strategy**, which iteratively adds and removes noise during training to simulate natural variability, like in [26].

Divide and Conquer

Divide and conquer paradigm breaks the task of video synthesis into smaller, more manageable tasks. This approach can further by divided into 3 methods of approach:

1. **Hierarchical architecture for frame generation**: in this approach, the process is split into **keyframe generation** and **frame filling** models. Keyframes represent the critical moments of the video that establish the narrative, while the frame-filling models ensure smooth transitions between them. Global models focus on generating the main storyline through keyframes, and local models handle finer details, filling the gaps between the keyframes to maintain temporal consistency.
2. **Multi-stage approach for video super-resolution**: [11] suggested generating video by stages, first by creating low-resolution sequences with GAN and then applying super-resolution models to enhance them with StyleGAN3 [42].

Evaluation Metrics	Category	Description	Discrepancy Dimension
FID(Heusel *et al.*, 2017)	Image/Frame level indicators	Evaluating the quality of a generated video by comparing synthesized video frames with real video frames.	Keyframes quality
PSNR(Huynh-Thu and Ghanbari, 2008)		Evaluating the ratio between the signal and noise in video frames.	Frames Signal-to-noise ratio
SSIM(Wang *et al.*, 2004)		Evaluating the structural similarity between original and generated images/frames, taking into account brightness, contrast, and structural similarity.	Frames structural similarity
CLIPSIM(Radford *et al.*, 2021)		Evaluating the correlation between images/frames and text.	Image and text correlation
LPIPS(Zhang *et al.*, 2018)		Evaluating the perceptual similarity between generated video frames and real video frames.	Frames perceptual similarity
FVD(Harvey *et al.*, 2022)	Video level indicators	Evaluating the similarity of the generated video distribution in feature space to the real video distribution.	Feature distribution
KVD(Unterthiner *et al.*, 2018)		Evaluating the quality of generated videos by using Maximum Mean Discrepancy (MMD).	Maximum Mean Discrepancy
IS(Salimans *et al.*, 2016)		Calculating the Inception score of generated videos using features extracted by 3D-ConvNets (C3D).	Inception scores

Figure 49: Evaluation metrics in video generation [47].

3. **Integrated keyframe and frame filling with mask modeling** simplifies long video generation by combining keyframe creation and frame filling into a unified process. Mask modeling hides specific parts of the video during training, keyframes and intermediate frames simultaneously, like done in CogVideo [37], which simplified mask modeling and combines these models into a single model.

Spatial autoregressive models

Spatial autoregressive models are adept at processing tokenized sequence inputs (they include the transformer architecture) which enable the segmentation of video frames into patches. By integrating video data features with the autoregressive power of transformers, these models become more capable of capturing temporal dynamics. In this method, there are two main approaches:

- **Frame tokenization**: like in NUWA-Infinity [101], where video frames are transformed into patches, and each patch is projected to a patch embedding, and combine them with spatial positional encodings. These autoregressive models, similar to LDMs, compress video into latent space. Compression techniques like **Discrete Cosine Transform (DCT)** are used in Transframer [53] framework and by using a VQ-GAN, they convert the frames into latent tokens.
- **Scaling with attention mechanism**: architectural modifications were made by incorporating specialized blocks, such as attention mechanism blocks tailored for spatiotemporal data. For example, Transframer [53] integrated temporal and spatial annotations (time-steps, camera viewport, etc.) through cross-attention.

10.1 Evaluation metrics

A common metric used for video generation models is the **Frechet Video Distance (FVD)** [94], which is an extension of Frechet Inception Distance (FID). FVD compares videos by both spatial and temporal features by using similar process as FID: using a pre-trained 3D ConvNet (i3D), where in FID they use InceptionV3. In similar fashion as FID, FVD compares the distribution of features of video clips:

Method	Input information	Task
GAN models		
VideoGAN [9]	Video	Video generation and video prediction given a single static image in a close-set scene domain.
EDN [15]	Video	Video-to-video translation using pose as an intermediate representation.
VAE models		
SVG [10]	Video	Video prediction given the initial frames of a simple motion video like human activity
SadTalker [16]	Image, audio	Talking head generation given a face image and a piece of speech audio.
Autoregressive models		
Video Pixel Networks [11]	Video	Video prediction given the initial frames of a simple motion video like MNIST motion.
CogVideo [17]	video, text	Text-to-video generation, video prediction, and video frame interpolation.
Diffusion models		
VDM [12]	Video, text/label	Text-conditioned or label-conditioned video generation, and video prediction.
Imagen-Video [13]	video, text	Text-to-video generation, video prediction, and video frame interpolation.
Make-a-Video [14]	video, text	Text-to-video generation, video prediction, and video frame interpolation.
Video LDM [18]	video, text	Text-to-video generation, high-Resolution video synthesis.
DreamTalk [19]	Image, audio	Talking head generation given a face image and a piece of speech audio.
Dancing Avatar [20]	Motion, text	Generating highquality human videos guided by textual descriptions and motion.
Discro [21]	Motion, text	Generating highquality human videos guided by textual descriptions and motion.

Figure 50: Video generation models & papers [112].

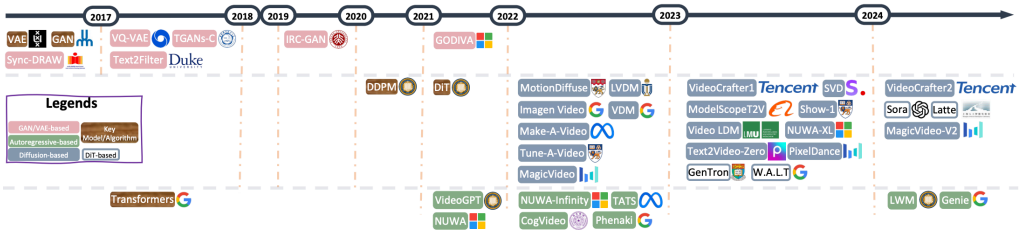


Figure 51: Text-to-Video generators evolutionary timeline, based on foundational algorithms [89].

$$\text{FVD} = ||\mu_{\text{real}} - \mu_{\text{fake}}||_2^2 + \text{Tr}\left(\Sigma_{\text{real}} + \Sigma_{\text{fake}} - 2\left(\Sigma_{\text{real}}\Sigma_{\text{fake}}\right)^{1/2}\right)$$

where μ_{real}, μ_{fake} is the mean of the real and fake data, Σ_{real}, Σ_{fake} is the covariance matrix of the real and fake data, Tr function is the trace of the matrix, and $||\cdot||_2^2$ is the squared Euclidean norm.

The lower the FVD score, the better the video generation model. A more comprehensive list of evaluation metrics is shown in figure 49.

In the Diffusion Transformer (DiT) paper [58] the researchers used **Gflops** (giga floating point operations per second) metric to measure the complexity of the model, instead of traditional parameter count of the model. Higher Gflops means the model is more scalable and complex.

10.2 Previous works

In figure 50 we can see different models, divided into four main categories (VAE, GAN, diffusion, autoregressive) for the task of video synthesis. Different models have different input methods (video, text, audio, motion, etc.).

In figure 51 we can see the timeline of different text-to-video models, their paper origin (Meta, Microsoft, Google, OpenAI, and even some universities) based on foundational algorithms (GAN/-VAE, Autoregressive, Diffusion and Diffusion Transformer). We can also see the key models/algorithms, such as transformers, DDPM, DiT, and more.

NUWA-Infinity [101], introduced by Microsoft AI in 2022, is a T2V model capable of generating videos of arbitrary resolution and length. It ensures spatial and temporal consistency through

an innovative **autoregressive over autoregressive** generation mechanism. This mechanism combines two levels of autoregressive modeling: a global patch-level model that captures dependencies between patches and a local token-level model that addresses dependencies within visual tokens of each patch.

NUWA-XL [107] (2023, which is an upgraded version of NUWA-Infinity) is a diffusion T2V model that uses a 3D U-Net in global model for keyframe generation and a model for keyframe filling. They also built a new dataset called 'FlintstonesHD' for benchmarking long video generation. Their method reduced inference time when generating 1024 frames from 7.55 minutes to 26 seconds (**a 94% reduction in inference time**) on the same hardware.

Image-GPT (iGPT): OpenAI's 2020 paper introduced Image-GPT [13], an autoregressive image generation transformer model that predicts pixels directly using attention mechanisms. Pre-trained on ImageNet, iGPT demonstrated the ability to learn spatial features for image representation.

Video Diffusion Models (VDM): Google's 2022 work [36] extended image diffusion models to video generation using a 3D U-Net architecture. They replaced 2D convolutions with space-only 3D convolutions and introduced temporal attention blocks alongside spatial ones. Relative position embeddings were used to capture frame order, enabling the model to handle both images and videos.

MagicVideo (2022) [111] from ByteDance is a T2V model leverage LDM based on 3D U-Net and a directed temporal attention, and a pre-trained VAE to map video clips to low-dimension latents to learn the distribution of the latent codes. It is based on keyframe and frame filling paradigm and uses CLIP to encode the text prompt.

CogVideo (2022) [37] is a 9 billion parameters open-source video synthesis model that is based on CogView2, which is a T2I model. It uses an autoregressive transformer, that works in latent space to generate video frames. CogVideo also lists VideoGPT as related work, and upgrades upon it.

EasyAnimate: Alibaba's 2024 multi-modal diffusion video generation model [103] uses a 12-billion-parameter Diffusion Transformer (DiT) (appendix G) to scale attention mechanisms to the video domain. It introduces Slice VAE, which reduces GPU memory usage compared to traditional video VAEs.

To summarize: Researchers agree that generating high-cohesion, long videos requires modeling long-range temporal dependencies. GANs are challenging to optimize for video tasks due to adversarial loss, unlike the likelihood loss used in VAEs, diffusion models, and transformers. And finally, they agree that:

> *"... the video domain has not yet witnessed its 'AlexNet moment'"* [92]

AlexNet [45] was a major breakthrough in 2012 for image classification task which led to the development of imaging domain in deep learning.

10.3 Spatiotemporal feature learning

One of the first work that tries to capture temporal dynamics is in a form of sequence-to-sequence video captioning [99] (2015). In this paper, the researchers used **Long Short-Term Memory model (LSTM)** to caption video, which was a popular model for sequence tasks at that time (and also Recurrent Neural Networks [RNN] which was prior work to LSTMs). However, LSTMs struggled with issues like exploding/vanishing gradients, leading to their replacement by more scalable models.

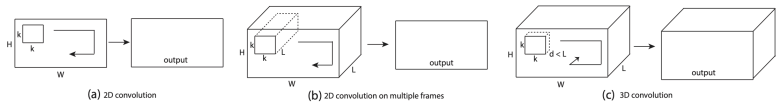


Figure 52: 2D and 3D convolution operations. (a) Applying 2D convolution on image results in an image. (b) Applying 2D convolution on video (multiple frames) also results in an image. (c) Applying 3D convolution on video results in another volume (channel), preserving temporal information [93].

Figure 53: Feature maps of the first 3D convolution layer of the C3D model [93]. The first two rows of the feature maps shows the detection of moving edges and blobs, while the last row shows edge orientation changes and color changes [93].

Modern approaches leverage 3D convolution networks (3D ConvNets) and temporal attention. A 2014 paper [93] by Facebook AI introduced 3D ConvNets for spatiotemporal feature learning. The researchers found that 3x3x3 kernel size is the best option for 3D ConvNets. After every convolution layer they apply 3D max-pooling.

Figure 53 shows the feature maps of the first 3D convolution layer of the C3D model [93].

And in figure 52 we can see how 3D convolution evolved from 1D and 2D convolution. The output of 2D convolution flat (2D), compared to 3D convolution where we also have volume, which is used as the frames dimension in video synthesis models.

Further improvements in a 2018 paper [92] also by Facebook AI proposed R(2+1)D blocks, separating spatial and temporal convolutions into a 2D residual block followed by a 1D temporal convolution. This structure enhanced optimization and enabled modeling of complex non-linear functions, making it a strong performer for action recognition.

10.4 Practices & techniques in vision domain

In this section we explore common practices or techniques used in vision domain, that are applied in image synthesis and video synthesis models.

Use of transformers for multi-modal outputs: Transformers have been shown time and time again their capability to scale very well given large enough datasets. In addition, they excel at sequence-to-sequence tasks, which allows researchers to use them in their vision models to condition the model on multiple types of data (text, audio, image, layouts, semantic masks, etc.).

Patchify: In Vision Transformer (ViT) [20] (appendix F) and Diffusion Transformer (DiT) [58] (appendix G) both use patch technique to model and represent patches of size $p \times p$ of an image. First the image is divided into patches, then each patch is linearly embedded into a vector, and the position (either absolute, relative or even distance between patches) of the patch is added (element-wise) to the patch embeddings, forming an embedding of patch + position tokens. These embeddings are then used in the transformer to perform vision task (generation, mask filling, etc.).

Spatial Self-Attention (SSA): Self-attention mechanism is often used to capture long-range spatial dependencies in a single frame. Unlike CNNs, which capture local features in their kernels, self-attention allows the model to learn the global structure of an image.

Temporal Self-Attention (TSA): Self-attention mechanism is also often used to capture temporal dependencies (relationships between frames). It enables the model to model how the video changes over time.

Color reduction: Image-GPT [13] proposed context reduction: first by downsizing the resolution of the training videos to lower resolution, and created their own 9-bit color palette by K-mean clustering (k=512). Each pixel is represented by 9 bits, instead of 24 bits ($256 \times 256 \times 256$ color options per pixel). This color reduction yields a sequence length three times shorter than the standard (R, G, B) palette.

Use of compute efficient attention blocks: Like in the case of Video-GPT [106] which uses axial attention (appendix D), which doesn't apply self-attention to the entire image, rather only to the current row and column, separately. This reduces the complexity of the attention mechanism, from $O(n^2)$ to $O(n \cdot \sqrt{n})$.

Operating in latent space: Because video synthesis is compute intensive task, a lot of previous works operate in latent space, which is compressed lower-dimension space, compared to raw RGB frames in the pixel space. Most of the time, VAEs are used to encode images to latent space and decoder to decode the latent space back to the pixel space.

Conditioning mechanisms: There are three ways to condition an image/video generation model: cross-attention, adaptive layer normalization, (like in diffusion transformer) and in transformer based models, the conditional information is concatenated to the input sequence tokens (like in ViT and DiT) to be fed into the transformer decoder.

Converting image generation model to video generation model: Video-LDM [8] for instance, used a pre-trained image generator and then converted it to video generator by adding temporal layers.

Reusing pre-trained models: Imagen [75] used a pre-trained T5-XXL text encoder to encode the text prompts, for example.

Inserting temporal layers: Inserting temporal layers into image synthesis models has been explored by MoCoGAN-HD [91], StyleVideoGAN [22] and Video-LDM [8]. This technique adds temporal layers to pre-trained image generation model, which converts the model from to video generator.

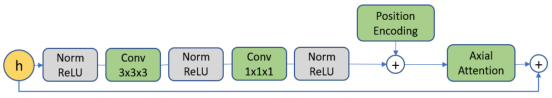


Figure 54: Residual attention block as described in VideoGPT [106] which are used in the VQ-VAE model [106].

11 VideoGPT

VideoGPT [106] uses likelihood based VQ-VAE model and a transformer model based on the GPT architecture. The encoder of VQ-VAE first downsamples video into discrete latent space using 3D convolutions, and then the decoder reconstructs the video from these latent codes.

The VQ-VAE

Reminder that the loss function of VQ-VAE is defined as:

$$\mathcal{L} = \underbrace{||x - D(e)||_2^2}_{\mathcal{L}_{\text{recon}}} + \underbrace{||\text{sg}\left[E(x)\right] - e||_2^2}_{\mathcal{L}_{\text{codebook}}} + \underbrace{\beta\,||\text{sg}[e] - E(x)||_2^2}_{\mathcal{L}_{\text{commit}}}$$

where sg refers to stop-gradient, $\mathcal{L}_{\text{recon}}$ is the **reconstruction loss** (encourages the VQ-VAE to learn good representations to accurately reconstruct the data), $\mathcal{L}_{\text{codebook}}$ is the **codebook loss** (quantization loss, it brings codebook vectors closer to the encoder outputs), and $\mathcal{L}_{\text{commit}}$ is the **commitment loss** (which prevents the encoder outputs from fluctuating between different code vectors).

The reconstruction loss optimizes only the decoder, the commitment loss optimizes only the encoder, and the codebook loss optimizes the codebook embeddings only. See section 4 for more details.

The researchers used EMA (exponential moving average which is described in the VQ-VAE paper [95]) update for the codebook loss. In short, it makes the VQ-VAE faster to train and converge.

To train the VQ-VAE model (to learn the latent codes), the authors trained the VQ-VAE on video data. The encoder consists of a downsample 3D convolutions followed by attention residual blocks (figure 54), and `LayerNorm`. The `LayerNorm` layers are described in appendix C.3. The position embeddings are not explained in the paper; but we can infer them as the frame position (i.e. frame 0, 1, ... in the video), which is added element-wise.

The architecture of the decoder is the reverse of the encoder (residual attention blocks followed by upscale 3D convolutions).

The transformer (GPT)

GPT is autoregressive transformer architecture (appendix E). The autoregressive nature can be described as:

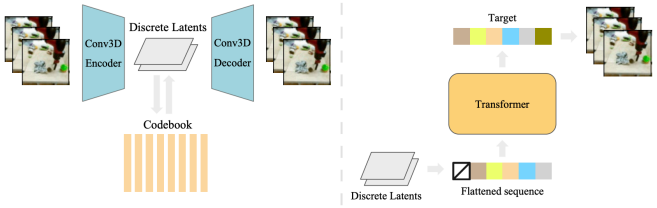


Figure 55: VideoGPT architecture [106]. Left: training the VQ-VAE; Right: training the autoregressive transformer model on the latent vectors [106].

$$p(x) = \prod_{i=1}^{d} p(x_i | x_{\leq i})$$

through masked self-attention. The GPT transformer is optimized using maximum likelihood (appendix A). The transformer follows the same prior networks as Image-GPT [13].

In **Image-GPT** the network learns to predict pixels and only a transformer is used, which is the first time a transformer-only image synthesis model was proposed (no CNN prior!). The model learns to predict pixels autoregressively based on a probability distribution modeled on each of the previous pixels. The attention mechanism helps in the modeling of the spatial pixel features. The iGPT-XL model consists of 6.8 billion parameters, iGPT-M has 455 million parameters, and iGPT-S has 76 million parameters. The researchers wanted to test how generative pre-training can be useful to learn image representations in large scale.

11.1 Architecture

The architecture of VideoGPT is shown in figure 55.

VQ-VAE: They train the VQ-VAE on video data to learn the codebook. The encoder and decoder consists of 3D convolutions (appendix C.4) followed by attention residual blocks (figure 54). Axial attention [33] block is described in appendix D. The purpose of the VQ-VAE is to encode the video into discrete latent space. The purpose of the transformer is to autoregressively generate the video from these latents.

Transformer: The right side of figure 55 shows the training of the autoregressive transformer model on the latent vectors. **The target is the next latent code (frame) in the video**. So the transformer learns to predict the next frame as a latent code.

11.2 Video generation

The model can start to autoregressively generate videos given starting image: the VQ-VAE turns the first image to latent vector, and this vector is then fed to the GPT transformer to generate the next frame. The process is repeated until the desired number of frames is generated.

Multiple video generation examples are shown in figure 56.

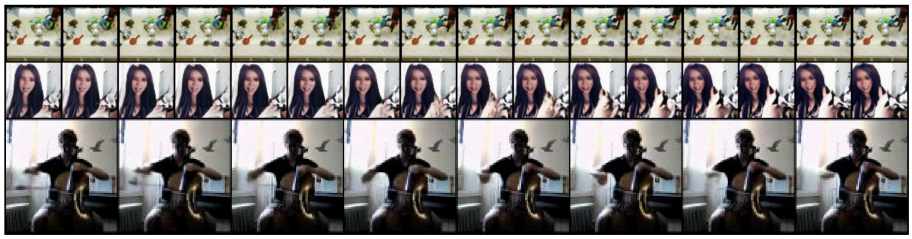


Figure 56: Video samples from VideoGPT [106].

12 Video-LDM

Video-LDM (2023) by NVIDIA [8] is a T2V model based on LDM. It first trains an image generator on image datasets, then freeze the layers, and then adds temporal layers which helps to align images in temporally consistent manner, transforming it from T2I model to T2V model. Then the model is fine-tuned on video dataset. Freezing the spatial layers transforms the image generation knowledge to the video domain. The model achieves high resolutions of up to 1280×2048.

Mathematical notations

The LDMs autoencoder compresses the high-dimensional input data $x \in \mathbb{R}^{T \times 3 \times \hat{H} \times \hat{W}}$ where $x \sim p_{\text{data}}$ is a sequence of T RGB frames with height $\hat{H}$ and width $\hat{W}$ to lower-dimensional latent space $z = \mathcal{E}(x) \in \mathbb{R}^{T \times C \times H \times W}$ (where C is the number of latent channels, and H, W are the spatial latent dimensions). The model then learns to reconstruct x with the decoder $\mathcal{D}$: $\hat{x} = \mathcal{D}(\mathcal{E}(x)) \sim x$.

Let us denote the spatial layers l_θ^i where θ are the model's parameters, and i is the layer index. We then denote the addition of temporal layers as l_ϕ^i to learn to align individual frames in a temporally consistent manner. We denote $f_{\theta,\phi}$ as the full model (with spatial and temporal layers).

12.1 Architecture

The architecture of Video-LDM is similar to Stable Diffusion except for the added temporal layers, and the addition of autoencoder for latent encoding/decoding and the discriminator which helps photorealism in the decoder network. The model's stack is shown in figure 57.

Figure 58 illustrates the stochastic diffusion process, showing SDE paths that represent sample trajectories. Data distribution $p(x)$ transitions to prior distribution $p_T(x)$ via forward SDE (adding noise) and reverses back to $p(x)$ using reverse SDE. As noise is added, structured data becomes progressively chaotic, converging to a Gaussian prior before reverting to the original distribution. The distribution depicts the data probability, the middle distribution depicts the Gaussian prior, and the right distributions depicts reconstructed data.

In figure 59 we see the temporal fine-tuning. We see the stochastic denoising process of diffusion, similar to figure 58. We can see that the images on the left side are not temporally aligned, whereas on the right, the images form a coherent video.

In figure 60 we can see the frozen encoder compresses video frames into 3D latents and the decoder learns to reconstruct them. On the right, we can see the patch-based discriminator $\mathcal{H}$ which is used to increase photorealism of the decoder. It is trained on the adversarial objective

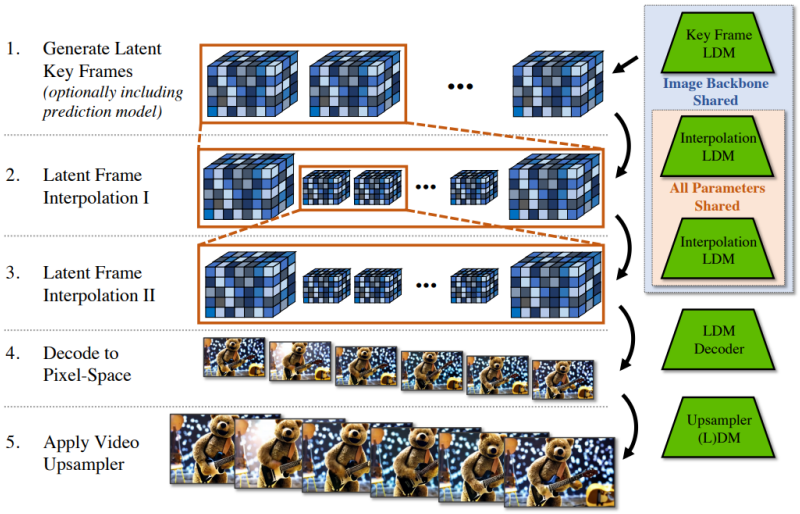


Figure 57: Video-LDM stack [8]. (1) Keyframe generation. (2) Keyframe filling (keyframe interpolation). (3) Keyframe interpolation is repeated a second time. (4) The latents are decoded to pixel space. (5) Optionally, SR model is applied [8].

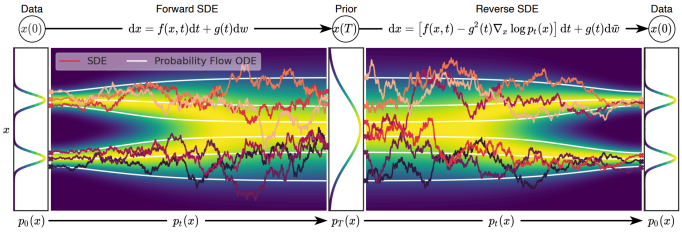


Figure 58: Generative modeling through stochastic differential equations (SDE) [86].

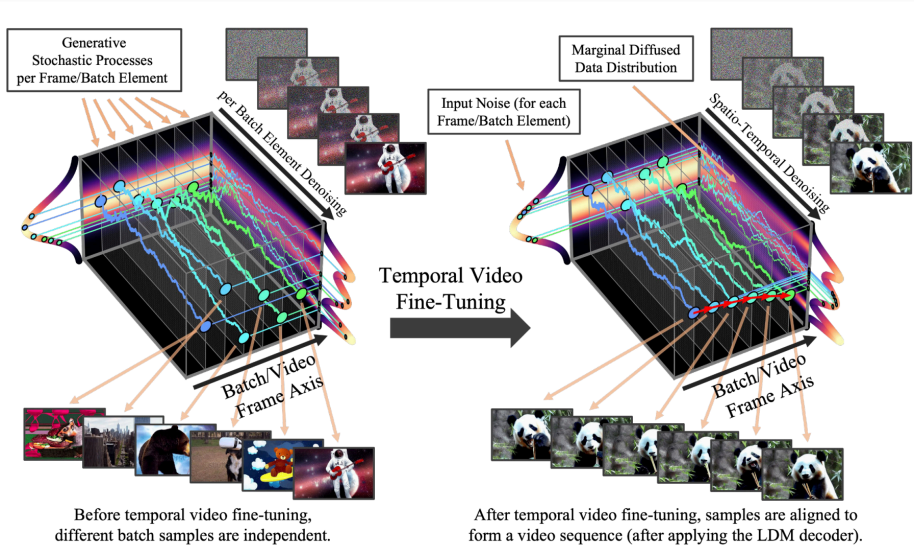


Figure 59: Transforming image LDM (left) to video LDM (right) via temporal fine-tuning [8].

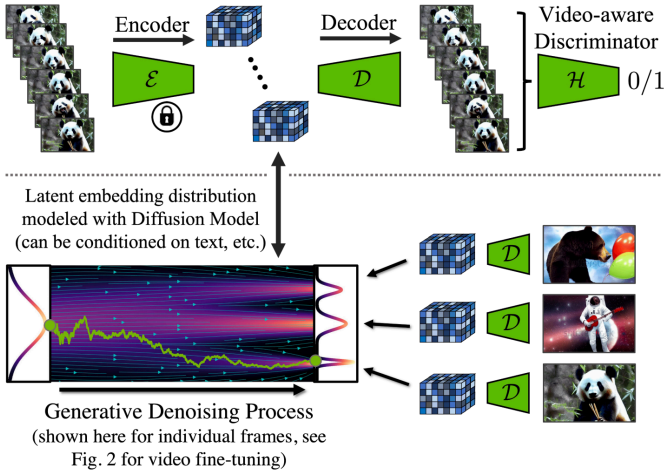


Figure 60: *Top*: frozen encoder *Right*: *Bottom*: generative denoising process. [8]

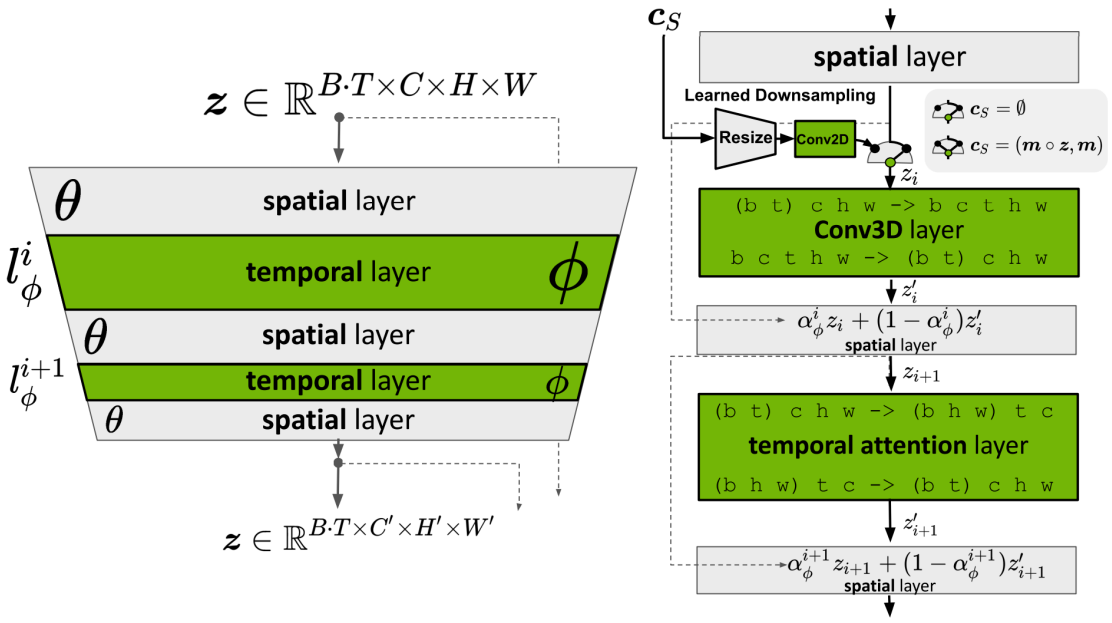


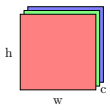
Figure 61: The researchers turn pre-trained LDM (in gray) into video generator by adding temporal layers (in green) [8].

which is added to the AE reconstruction score like in PatchGAN [39] (which tries to classify if $N \times N$ patch is real or fake). On the bottom we can see the generative denoising process, where each embedding corresponds to different image latent in the decoder $\mathcal{D}$.

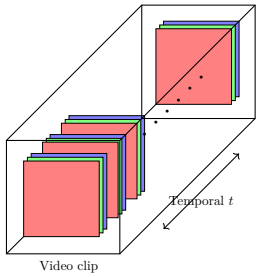
In figure 61 we can see that the pre-trained LDM (in gray) is turned into a video generator (on the right) by adding temporal layers l_ϕ (in green) that learn to align images in a temporally consistent manner. The spatial layers l_θ are frozen (in gray) and only the temporal layers are learned (in green). On the right we can see closer look of spatial, temporal layers. One problem arises, though. The pre-trained LDM is trained only on batch, channel, height and width dimensions, but video has one additional dimension: the temporal t dimension. So, in order to deal with the addition of the temporal dimension, they **rearranged the batch dimension** to be mixed with the temporal dimension (instead of b the batch dimension is now $(b\ t)$) for the next layer: $(b\ t)\ c\ h\ w \rightarrow b\ t\ c\ h\ w$. The c_S represents a context vector used for conditioning the model (explained below in subsection **context guidance & masking**).

The input of the spatial layers l^i_θ is of the dimension $b\ c\ h\ w$ whereas the input dimension of temporal layers is of the dimension $b\ t\ c\ h\ w$. This notation is called `Einops` (einstein operations) and is further discussed in the Einops paper [70]. The spatial layers interpret the video as a batch of images by shifting the temporal axis into the batch dimension (the spatial layers treat all $B \cdot T$ encoded video frames in the batch dimension b). It is important to note that the rearrange operation doesn't change the tensor values; rather it rearranges the dimensions.

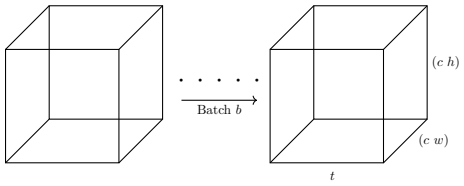
For the 3D convolution (temporal layer), they reshape the input back to video dimensions (they process entire videos in new temporal dimension t):



(a) A single frame. The $c\ h\ w$ dimensions.



(b) Multiple frames. The $t\ c\ h\ w$ dimensions.



(c) All the $b\ t\ c\ h\ w$ dimensions.

Figure 62: Representation of video dimensions $b\ t\ c\ h\ w$ which corresponds to batch, temporal, channel, height, width dimensions.

$$\underbrace{(b\ t)\ c\ h\ w}_{\text{Output of spatial layer}} \rightarrow \underbrace{b\ c\ t\ h\ w}_{\text{Input to the next Conv3D layer}}$$

After the temporal layers they reshape the input back to fit into the spatial layers like so:

$$\underbrace{b\ c\ t\ h\ w}_{\text{Output of Conv3D layer}} \rightarrow \underbrace{(b\ t)\ c\ h\ w}_{\text{Input to the next spatial layer}}$$

As for the temporal attention layer (temporal layer), they reshaped the input back to work on the temporal dimension, instead of spatial:

$$\underbrace{(b\ t)\ c\ h\ w}_{\text{Output of spatial layer}} \rightarrow \underbrace{(b\ h\ w)\ t\ c}_{\text{Input to the next temporal attention layer}}$$

$$\underbrace{(b\ h\ w)\ t\ c}_{\text{Output of temporal attention layer}} \rightarrow \underbrace{(b\ t)\ c\ h\ w}_{\text{Input to the next spatial layer}}$$

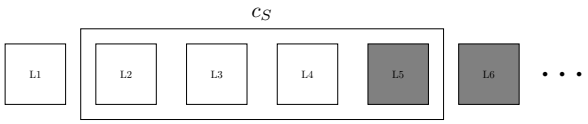


Figure 63: Keyframe model learns to predict the next latent frame z by context guidance (conditioned on previous frames).

We can see visual representation of the dimensions (batch, temporal, channel, height, width) in figure 62. It's made clear what each dimension represents in the video data.

Mixing factor: after each temporal layer, the output z' is combined with the output of previous spatial layer output z to form a mixing:

$$\underbrace{\alpha_\phi^i z_i + (1 - \alpha_\phi^i) z_i'}_{\text{Mixing factor}}$$

where $\alpha_\phi^i \in [0, 1]$ and is a learnable parameter. If we set $\alpha = 1$ for each layer skip the temporal score, and we retain the native image generation capability. This mixing operation is similar to CFG (mixing between conditional score and unconditional score).

Temporal mixing layers: two types of temporal mixing layers in use (see figure 61) are the *temporal attention* and *Conv3D* layer.

Noise scheduler: Video-LDM uses the same noise scheduler as the underlying image model. In table 6 of the paper, it shows that in all of their models, they use linear noise scheduler.

Training objective: in the training phase only the temporal layers are trained. The objective is the LDM objective (likelihood based, predicting noise in U-Net):

$$\arg\min_\phi \mathbb{E}_{x \sim p_{\text{data}}, \tau \sim p_\tau, \epsilon \sim \mathcal{N}(0,I)} \left[||y - f_{\theta,\phi}(z_\tau; c, \tau)||_2^2 \right]$$

where τ is the diffusion time step, y is the target noise vector. The target is to minimize the difference between predicted noise and ground truth y over all video frames.

Adding temporal layers to the decoder: the researchers add temporal layers to the decoder of the AE which they found this step to be critical for achieving good results. The reason is that the AE is trained on images and flickering artifacts are present in the generated videos (because training on images doesn't teach the model temporal dynamics). So the researchers fine-tune the decoder on video data with a **patch-wise temporal discriminator built from 3D convolutions**. The encoder, however, remains unchanged.

Patch-wise temporal discriminator: The patch-wise temporal discriminator $\mathcal{H}$, which is built from 3D convolutions, is used to fine-tune the decoder. It's trained on video data and returns "real" or "fake" prediction on patches of the video clip.

Context guidance & Masking: the model can be conditioned on context information, and is denoted with c_S in figure 61. The model is conditioned on the initial set of S context frames, which are the frames at the beginning of the clip. A temporal binary mask is applied to the frames that the model must predict, so the model knows which frames it must predict and which are the context. The latent vector generated by the encoder is concatenated with the binary mask, forming the context guidance c_S. In figures 63 and 64 we can see the context guidance in action: in the keyframe model and the interpolation model.

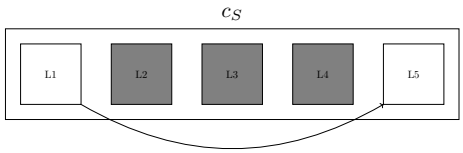


Figure 64: Interpolation model learns to fill masked latents $m \circ z$ (L2, L3, L4) in between two keyframes latents (L1, L5) with context guidance.

Temporal binary mask & Higher FPS: in order to adapt to long duration videos, the researchers trained a $T \rightarrow 4T$ keyframe interpolation model, where for each two keyframes, the model interpolates between them and generates 3 additional frames by using binary masks m_S (0 or 1). The frames are multiplied by the mask and the model learns to predict the missing (masked) frames. This technique achieves higher FPS.

LDM Decoder: the latents are then decoded to the pixel space using the LDM decoder.

Upsampler: to increase the spatial resolution of generated frames, they used a SR model, inspired by cascaded DMs [34] (the SR3 model [74]). In their experiments they used **pixel-space DM upsampler**, whereas for T2V models they used **LDM upsampler**. Since upsampling video frames independently would result in poor temporal consistency, they made the SR model video-aware by adding temporal layers and mixing spatial and temporal layers, in similar manner as discussed before.

Number of parameters: the Video-LDM model (except for the CLIP text encoder) consists of 3.1 billion parameters (AE and diffusion models [keyframe generation model, interpolation model, upsamplers]), and only 2.2 billion of these parameters are actually trained:

- 84 million parameters in the autoencoder
- 865 million parameters in the image backbone LDM, not including CLIP text encoder
- 655 million parameters in temporal layers
- 354 million parameters in the text encoder (OpenCLIP-ViT/H)
- 1.5 million parameters in the interpolation latent diffusion model

Compared to Imagen (11.6 billion parameters) and CogVideo (9 billion parameters), Video-LDM is much smaller yet produced high quality videos partly because it works in latent space.

DDIM Sampling: in appendix F of the paper they write that they use the DDIM sampler [84] (section 8.7), where the stochastically η is varied, as well as the guidance scale.

CLIP text encoder: the text encoder (for T2V conditioning) is a CLIP [63] based model which is used to generate text embeddings, as we discussed before (section 8.6).

12.2 Experiments

In figure 65 we see comparison of Video-LDM and Long Video GAN (LVG) on RDS dataset. `Cond.` means the model was conditioned on day/night and crowdedness[¶¶]. On the right we see FVD and

[¶¶]In their experiments, they observed that adding conditional information reduces FID and FVD metrics.

Method	FVD	FID
LVG [6]	478	53.5
Ours	389	**31.6**
Ours (cond.)	**356**	51.9

Method	FVD	FID
Pixel-baseline	639,56	59.70
End-to-end LDM	1155.10	71.26
Attention-only	704.41	50.01
Ours	534.17	**48.26**
Ours (context-guided)	**508.82**	54.16

Figure 65: Comparison of Video-LDM and Long Video GAN (LVG) on RDS dataset [8].

Figure 66: Real driving videos (RDS) generation samples [8].

FID evaluations on different diffusion architectures (pixel-space diffusion model, end-to-end LDM that was not pre-trained on images [which is why the results are bad], and attention-only temporal model). Their model uses 3D temporal convolutions which is better than the attention-only diffusion model.

In figure 66 we see real driving videos (RDS) samples. *Top*: night videos. *Middle*: the left red frame is the condition input, while the right is two different generated samples (given a frame, the model can generate the next frames). *Bottom*: they trained a separate bounding-box conditioned LDM for image synthesis only (the RDS dataset has some bounding-box clips), which generates initial frame (in yellow), and then Video-LDM completes the video generation based on this frame.

We can see text conditioned samples in figure 67 on the WebVid-10M dataset at 1280×2048 resolution.

In figure 68 we can see comparison of metrics on UCF-101 and MSR-VTT datasets between Video-LDM and other methods. Video-LDM achieves the best inception score on UCF-101 dataset, and almost achieves the best CLIP-SIM score on MSR-VTT dataset.

The researchers used the following datasets:

- **RDS (Real Driving Videos)**: An in-house dataset of 683,060 videos (up to 8 seconds @

Figure 67: Generated samples trained on WebVid-10M dataset. Prompts: "An astronaut flying in space, 4k, high resolution" and "Milk dripping into a cup of coffee, high definition, 4k" [8].

Method	Zero-Shot	IS (↑)	FVD (↓)
CogVideo (Chinese) [32]	Yes	23.55	751.34
CogVideo (English) [32]	Yes	25.27	701.59
MagicVideo [109]	Yes	-	699.00
Make-A-Video [76]	Yes	33.00	367.23
Video LDM *(Ours)*	Yes	33.45	550.61

(a) UCF-101 dataset. Video-LDM achieves the best inception score.

Method	Zero-Shot	CLIPSIM (↑)
GODIVA [98]	No	0.2402
NÜWA [99]	No	0.2439
CogVideo (Chinese) [32]	Yes	0.2614
CogVideo (English) [32]	Yes	0.2631
Make-A-Video [76]	Yes	0.3049
Video LDM *(Ours)*	Yes	0.2929

(b) MSR-VTT dataset. Video-LDM almost achieves the best CLIP-SIM score.

Figure 68: Video-LDM is compared to other state-of-the-art models in zero-shot setting on UCF-101 and MSR-VTT datasets [8].

512×1024 @ 30 fps) with day/night labels and "crowdedness" annotations. Some include car bounding boxes. They used label dropout for CFG and compared Video-LDM to Long Video GAN (LVG) on this dataset (figure 65).

- **WebVid-10M**: Contains 10.7M video-caption pairs (52K hours total). They fine-tuned Stable Diffusion spatial layers on frames, added temporal layers, and trained with captions. Upsampled videos to 1280×2048 resolution. Samples are 113 frames (4.7s @ 24 fps or 3.8s @ 30 fps).
- **Mountain Bike Dataset** [11]: Introduced by LVG, it contains 1,202 clips ($\geq$ 5 seconds @ 30 fps).

Evaluation metrics: they used FID to evaluate quality of individual frames. They also used FVD and human evaluation on video clips and compared to LVG. The human evaluation contained 100 videos @ 4 seconds; given a pair of videos (one from Video-LDM and one from LVG), participants select the most favorable.

Video synthesis on RDS dataset: for video synthesis on RDS dataset, they first generate a single frame using the image LDM, then they run the prediction model on a single frame and generated a sequence of keyframes. Next they performed two steps of the interpolation model which increases the FPS from 1.875 to 7.5 and from 7.5 to 30 fps. And then they ran the temporal upsampler which is run over portions of 8 video frames (the upsampler is also temporally aligned). In their experiments, the researchers successfully generated long, temporally coherent, high-resolution driving videos (up to 5 minutes), trained on the RDS dataset.

13 Imagen-Video

Imagen-Video by Google [35] is a T2V cascading diffusion model, based on Imagen (section 9). Imagen-Video has 7 sub-models in a cascading pipeline (figure 70). It generates high definition

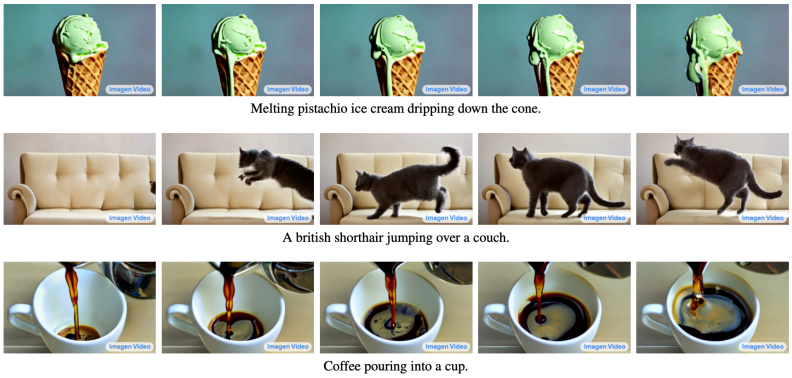


Figure 69: Imagen-Video video samples [35].

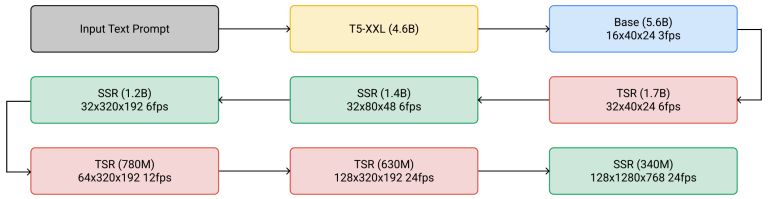


Figure 70: The cascading pipeline of Imagen-Video. The text embeddings are injected to all models in the pipeline (not shown) [35].

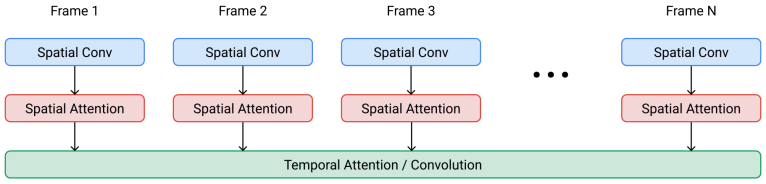


Figure 71: Video U-Net block [36] used by Imagen-Video [35].

1280×768 videos @ 24 fps (see figure 69 for samples), for 5.3 seconds. A big downside compared to Video-LDM is that Imagen-Video works in the pixel-space, whereas Video-LDM works in latent space. In addition, Imagen-Video is a much larger model and uses more compute resources than Video-LDM, however Google is able to achieve good scaling results through training on massive datasets.

Like Imagen, Imagen-Video uses the same large frozen text-encoder T5-XXL (section 9.1) in it is pipeline. The benefit of cascading pipeline is the ability to independently train each model, allowing the parallel training of all 7 models.

Imagen Video also builds on the work of **Video U-Net** [36], which generalizes the 2D diffusion model architecture to 3D in space-time by using temporal attention and 3D convolution layers to capture dependencies between video frames. See figure 71.

Due to safety and ethical concerns in training the Imagen-Video model, the researchers **decided not to release the model to the public** and keep it closed source.

One of the contribution of the paper is that they successfully transferred multiple methods from the image domain to video, such as **v-parameterization [76], CFG and conditioning augmentation** [34] (similar to Imagen). They also found out that **progressive distillation** [76] [49] is a valuable technique for speeding up video sampling.

13.1 Architecture & Method

Imagen Video has one frozen text encoder (T5-XXL), one base video diffusion model, three SSR (spatial super-resolution) models, and three TSR (temporal super-resolution) models - totaling 7 video diffusion models. Each of the denoising diffusion models $\hat{x_\theta}$ operate on multiple video frames simultaneously. In figure 71 the spatial convolution and spatial attention operate on each frame independently, while temporal attention and convolution operate on a collection of frames.

Whereas typically diffusion models for image generation use a 2D U-Net architecture (spatial attention and convolution), **Video U-Net** block used by Imagen-Video (figure 71) generalizes the U-Net to 3D space-time by using temporal attention and convolution to capture dependencies between video frames.

Imagen-Video pipeline is compromised of the following model types:

- **T5-XXL**: T5-XXL is a text-to-text transformer used to encode the text prompts to embeddings which are then used to condition all of the 7 diffusion models.
- **Base**: The base diffusion model generate low FPS low resolution video clip.
- **SSR**: The SSR model is a spatial SR (SSR) model that increases the resolution of the input image. Unlike the base model, it uses temporal convolution instead of temporal attention.

Like SR3, we first apply bilinear or bicubic interpolation to increase the resolution, then we remove noise (add details) by first concatenating noise y_t to the upsampled image x and then use the U-Net denoising network to learn to denoise the image.

- **TSR**: The TSR model is a temporal SR (TSR) model that increases the temporal resolution of the input video (increases frame count) by frame interpolation. This model also uses temporal convolution instead of temporal attention.

Temporal Attention v.s. Temporal Convolution: Temporal convolution reduces memory and computation costs over temporal attention, which is critical for the SSR and TSR models, because they are applied to high-resolution high-fps videos. This is why temporal attention is used at the beginning of the cascading pipeline.

Spatial attention at the beginning of the pipeline: The base model and the first two SSR models have spatial attention in addition to spatial convolution, because spatial attention at the beginning of the pipeline requires less compute resources than at the end. The last SSR model in the pipeline is a fully convolutional model (without attention).

Number of parameters: Imagen-Video consists of 11.6 billion parameters. Each of the model's parameters count is shown in figure 70.

Classifier-free guidance: CFG is also used in Imagen-Video. They found that it helps the model generate high fidelity samples with respect to the text prompts. Higher guidance weights lead the model to focus more on the text prompt conditioning.

Oscillating guidance: Similar to Imagen, when the guidance weight is too large, the possible range of values of predicted noise is beyond $[-1, 1]$, which causes train-test mismatch. This leads to significant artifacts in the generated videos. **Dynamic thresholding**, as described in section 9.3, helps to prevent this issue, which dynamically clip the image to the chosen threshold followed by scaling by s: `np.clip(x, -s, s) / s`. However, a constant high guidance weight leads to saturation artifacts, especially at high resolutions. The researchers didn't find dynamic thresholding sufficient, and they experimented with letting the guidance weight **oscillate between high and low values** (min and max) for a certain number of sampling steps. They apply this **oscillating dynamic threshold** only to the base and first SR models. The generation starts with a high guidance weight to establish a strong alignment with the prompt, then alternates between high and low weights in subsequent steps.

Noise conditioning augmentation: Similar to Imagen, Imagen-Video applied noise conditioning augmentation for all the spatial and temporal diffusion models. Noise conditioning augmentation is used to **corrupt low-resolution images**, and then the **SR model would be conditioned on the noise level**. The corruption noise level ("**augmentation level**") helps the model to generalize better to various noise levels and improves the diversity and quality of samples. This technique was introduced in a 2021 paper [34] by Google.

13.2 Video-image joint training

Training on both images and videos improves fidelity and enables knowledge transfer from images to videos, addressing the scarcity of text-video pairs datasets. This approach allows the model to learn diverse video styles.

Imagen-Video adopts the joint training method from [36], treating individual images as single-frame videos. To exclude temporal components during image training, they mask the temporal computation paths, preventing temporal operations across frames.

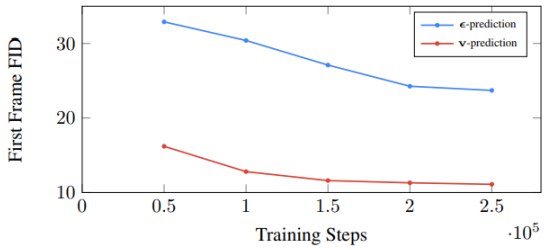


Figure 72: ϵ-prediction v.s. v-prediction parameterization. v-prediction converges faster [35].

13.3 v-prediction

v-prediction (velocity prediction) is a parameterization strategy used in diffusion models to predict the change of pixels in time of the diffusion process. Unlike ϵ-prediction, which the U-Net predicts the noise, v-prediction predicts the change of noise. There is also x-prediction in which the U-Net predicts the clean, original data x_t at timestep t instead of the noise added.

v-prediction is defined as:

$$v \equiv \alpha_t \epsilon - \sigma_t x$$

where α_t is the time-dependent coefficient determined by noise scheduler, ϵ is the noise, x is the data, and σ_t is the standard deviation of the noise at timestep t.

In figure 72 we can see that v-prediction requires less training steps to get lower first frame FID score (they only measured the FID score of the first frame of the generated video) compared to ϵ-prediction, because v-prediction is more stable and converges faster than ϵ-prediction. v-prediction is used in Imagen-Video to predict the change of pixels in time of the diffusion process, which helps the model to generate high-quality videos.

13.4 Progressive distillation with guidance and stochastic samplers

In a 2022 paper by Google [76], the team introduced **progressive distillation**, a fast sampling method for diffusion models. Unlike traditional DDPM samplers, requiring thousands of steps, progressive distillation enables sampling in a fixed, reduced number of steps. By iteratively distilling a DDIM sampler (see section 8.7), the required steps are halved with each distillation. This method is used with v-prediction.

In figure 73 we see the distillation process: instead of taking 4 sampler steps ($f(z;\eta)$) to transform noise ϵ to an image x, the distilled model now takes a single step.

In progressive distillation we duplicate a model that we desire to distill, which becomes a "student" and "teacher" model (they start with the same weights and biases). The student model learns to produce the output of the teacher model in fewer steps. The teacher model uses two steps of DDIM sampling and the student should learn to output the same teacher's output but in a single sampling step.

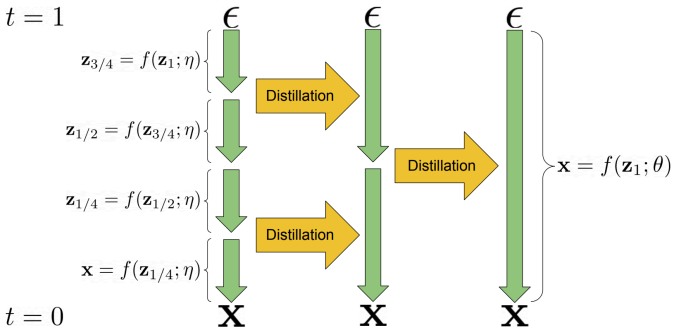


Figure 73: Progressive distillation process [76].

Algorithm 3 Standard diffusion training algorithm [76]

Require: Model $\hat{\mathbf{x}}_\theta(\mathbf{z}_t)$ to be trained
Require: Data set $\mathcal{D}$
Require: Loss weight function $w()$

while not converged **do**
  $\mathbf{x} \sim \mathcal{D}$ ▷ Sample data
  $t \sim U[0, 1]$ ▷ Sample time
  $\epsilon \sim \mathcal{N}(0, I)$ ▷ Sample noise
  $\mathbf{z}_t = \alpha_t \mathbf{x} + \sigma_t \epsilon$ ▷ Add noise to data

  $\tilde{\mathbf{x}} = \mathbf{x}$ ▷ Clean data is target for $\hat{\mathbf{x}}$
  $\lambda_t = \log[\alpha_t^2 / \sigma_t^2]$ ▷ log-SNR
  $L_\theta = w(\lambda_t) \|\tilde{\mathbf{x}} - \hat{\mathbf{x}}_\theta(\mathbf{z}_t)\|_2^2$ ▷ Loss
  $\theta \leftarrow \theta - \gamma \nabla_\theta L_\theta$ ▷ Optimization
end while

Algorithm 4 Progressive distillation algorithm [76]

Require: Trained teacher model $\hat{\mathbf{x}}_\eta(\mathbf{z}_t)$
Require: Data set $\mathcal{D}$
Require: Loss weight function $w()$
Require: Student sampling steps N

for K iterations **do**
  $\theta \leftarrow \eta$ ▷ Init student from teacher
  while not converged **do**
    $\mathbf{x} \sim \mathcal{D}$
    $t = i/N, \quad i \sim \text{Cat}[1, 2, \ldots, N]$
    $\epsilon \sim \mathcal{N}(0, I)$
    $\mathbf{z}_t = \alpha_t \mathbf{x} + \sigma_t \epsilon$
    # 2 steps of DDIM with teacher
    $t' = t - 0.5/N, \quad t'' = t - 1/N$
    $\mathbf{z}_{t'} = \alpha_{t'} \hat{\mathbf{x}}_\eta(\mathbf{z}_t) + \frac{\sigma_{t'}}{\sigma_t}(\mathbf{z}_t - \alpha_t \hat{\mathbf{x}}_\eta(\mathbf{z}_t))$
    $\mathbf{z}_{t''} = \alpha_{t''} \hat{\mathbf{x}}_\eta(\mathbf{z}_{t'}) + \frac{\sigma_{t''}}{\sigma_{t'}}(\mathbf{z}_{t'} - \alpha_{t'} \hat{\mathbf{x}}_\eta(\mathbf{z}_{t'}))$
    $\tilde{\mathbf{x}} = \frac{\mathbf{z}_{t''} - (\sigma_{t''}/\sigma_t)\mathbf{z}_t}{\alpha_{t''} - (\sigma_{t''}/\sigma_t)\alpha_t}$ ▷ Teacher $\hat{\mathbf{x}}$ target
    $\lambda_t = \log[\alpha_t^2 / \sigma_t^2]$
    $L_\theta = w(\lambda_t) \|\tilde{\mathbf{x}} - \hat{\mathbf{x}}_\theta(\mathbf{z}_t)\|_2^2$
    $\theta \leftarrow \theta - \gamma \nabla_\theta L_\theta$
  end while
  $\eta \leftarrow \theta$ ▷ Student becomes next teacher
  $N \leftarrow N/2$ ▷ Halve number of sampling steps
end for

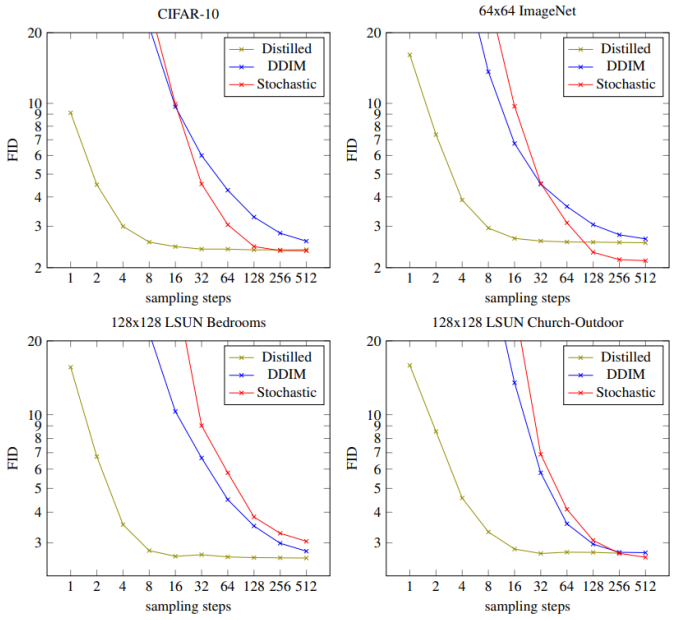


Figure 74: Comparison of v-prediction sampling to DDIM and stochastic samplers. The distilled model achieves significant lower FID scores with significantly fewer sampling steps (converges faster) [76].

This process is illustrated in algorithms 4 and 3. In algorithm 4 the marked lines are the only modifications made to the original training algorithm 3 of diffusion models. The teacher model $\hat{x}_\eta(z_t)$ is the diffusion model $\hat{x}_\theta(z_t)$ where z_t is the noisy data (we take training data $\mathcal{D}$ and add noise to it). After the student model (with parameters η) learns to predict $\tilde{x}$ (which is two DDIM steps), is distilled to match the output of the teacher model. Then the teacher model becomes the student and the process is repeated.

We can see in figure 74 that using a distilled model, sampling takes significantly fewer steps to converge to the same quality as the DDIM or stochastic (DDPM) samplers.

At higher resolutions, v-prediction avoids temporal color shifting artifacts compared to ϵ-prediction. We can see that in figure 75. The researchers say the reason is that in distillation we have to work at higher noise levels to the point where the signal-to-noise ratio (SNR) drops to zero, at which point predicting the noise becomes a lot harder.

Distillation with classifier-free guidance

The paper [49] by Google Research and Stability AI extends [76] by applying distillation to models trained with CFG, marking it is first use in video domain. In this work, the teacher model (trained with CFG, before distillation) takes two DDIM steps, and the student model, denoted in the paper as a w-conditioned model, learns to match the teacher's output while being conditioned on the guidance scale. This conditioning incorporates the guidance weight into the model backbone, similar to timestep conditioning in [43]. The student model is optimized with the following objective

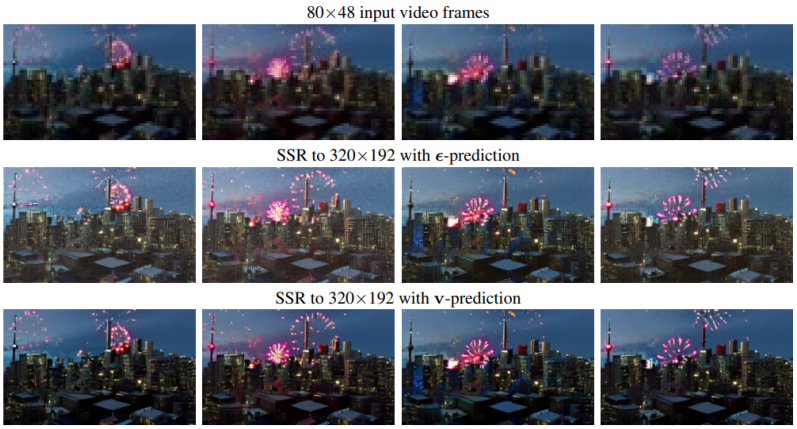


Figure 75: ϵ-prediction (middle row) v.s. v-prediction (bottom row). In ϵ-prediction we see color shifts across frames, whereas v-prediction is more consistent [35].

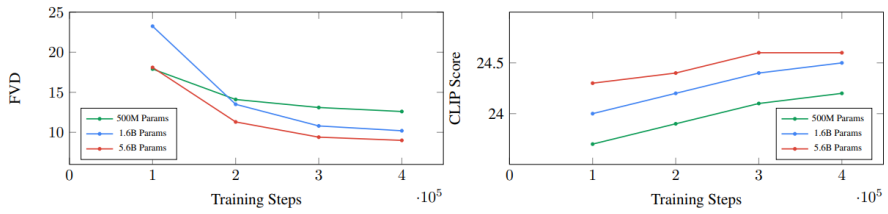
[49]:

$$\mathbb{E}_{w\sim p_w, t\sim U[0,1], x\sim p_{\text{data}}(x)}\left[\omega(\lambda_t)\left\|\underbrace{\hat{x}_{\eta_1}(z_t, w)}_{\text{student's pred}} - \underbrace{\hat{x}^w_\theta(z_t)}_{\text{teacher's pred}}\right\|_2^2\right]$$

where:

- w is the guidance weight and is picked from p_w: $w \sim p_w$ (which can be fixed or oscillate, but in Imagen-Video they chose to oscillate)
- $p_w(w) = U[w_{\min}, w_{\max}]$ is the oscillating guidance weight function
- t is the diffusion timestep
- x is the data (images): $x \sim p_{\text{data}}(x)$
- z_t is the noised version of x (noisy image at timestep t)
- λ_t is the guidance scale (not really important)
- $\omega(\lambda_t)$ is the weighting function (also not really important)
- $\hat{x}_{\eta_1}$ is the student model (which is conditioned on the guidance weight w as input: $\hat{x}_{\eta_1}(z_t, w)$). Also notice the notation: $\hat{x}_{\eta_1}$, which means that this is the first distillation iteration.
- $\hat{x}^w_\theta$ is the teacher model. Notice in the notation, it is using the guidance weight in CFG (it is not given as input): $\hat{x}^{\mathbf{w}}_\theta(z_t)$

Also notice that the teacher model's parameters (η) are different to the student model's parameters (θ), since they change at each distillation step (the student is a copy of the teacher but to



(a) Scaling Comparison on FVD scores.

(b) Scaling Comparison on CLIP scores.

Figure 76: Imagen-Video scaling results on video U-Net. The more parameters the model has, the better the FVD and CLIP scores [35].

differentiate between the parameters after the distillation process they chose to differentiate the parameters notations).

In the [49] paper we optimize the student's objective:

$$\hat{x}_\theta^w(z_t) = \underbrace{(1+w)\hat{x}_{c,\theta}(z_t)}_{\text{conditional score}} - \underbrace{w\hat{x}_\theta(z_t)}_{\text{unconditional score}}$$

where c is the teacher model's conditioning (for example, text prompt)[***].

To summarize, Using the work of [49], Imagen-Video researchers successfully distilled 7 video diffusion models with CFG to just 8 sampling steps without any noticeable loss in perceptual quality in the pipeline.

13.5 Experiments

In their experiments (in the Imagen-Video paper [35]), they sampled video clips at 192×320 resolution @[†††] 24 fps for 128 frames (5.3 seconds). They repeat the evaluations over four runs and report the mean and standard error.

Datasets: The researchers used 14 billion video-text pairs and 60 million image-text pairs from internal and public datasets, such as LAION-400M [79].

Pre-processing: They resized images using antialiased bilinear resizing and temporally resize videos by skipping frames.

Evaluation: They used FID on individual video frames, FVD for video, and frame-wise CLIP scores for video-text alignment (they take the average across all frames).

Scaling: In figure 76 we can see an interesting result which contradicts the findings of Imagen [75] paper. They found that increasing the U-Net size doesn't scale the sample quality as much as increasing the text encoder size. They conclude that video modeling task for which the performance is not yet saturated at current model sizes[‡‡‡].

[***]Thanks to this youtube tutorial for explaining both papers

[†††]In the paper they used the symbol '@' to say 'at'. For instance: "4.2s @ 24 fps" means "4.2 seconds at 24 frames per second".

[‡‡‡]This finding could hint that transformers, which benefit from scaling on large datasets, could provide the necessary scaling capabilities that video modeling task demands.

Guidance w	Base Steps	SR Steps	CLIP Score	CLIP R-Precision	Sampling Time
constant=6	256	128	25.19±.03	92.12±.53	618 sec
oscillate(15,1)	256	128	25.02±.08	89.91±.96	618 sec
constant=6	256	8	25.29±.05	90.88±.50	135 sec
oscillate(15,1)	256	8	25.15±.09	88.78±.69	135 sec
constant=6	8	8	25.03±.05	89.68±.38	35 sec
oscillate(15,1)	8	8	25.12±.07	90.97±.46	35 sec
ground truth			24.27	86.18	

Figure 77: Testing of fixed and oscillating guidance weight, distillation of base and SR models and evaluating each pipeline on CLIP score and sampling time. Distilling the pipeline results in significantly faster sampling time, while oscillating guidance weight improves the CLIP score a little [35].

Quality vs. Sampling time in a distilled model: In figure 77 we can see that distillation provides a good trade-off between sampling time and quality. Distilled model is 18 times faster in sampling, without significant degradation in perceptual quality. In addition, the distilled model is also 36 times more efficient in terms of compute cost in FLOPs metric.

14 Make-a-Video

Make-a-Video [82] (2022) by Meta AI is a T2V model that extends T2I capabilities to video generation using a spatiotemporally factorized diffusion model. Unlike many approaches, it doesn't require paired text-video data, enabling unsupervised training on large video datasets.

The model employs spatial and temporal SR strategies to generate high-resolution, high-frame-rate videos from text prompts. It also leverages image priors to simplify the complex task of video modeling. However, it cannot learn video-specific associations (e.g., directional hand movement) due to its reliance on unlabeled video data.

The model has 3 main stages:

- Training a T2I model on text-image pairs (based on DALL-E 2 [66] by OpenAI) (section 14.2.1).
- Adapting to T2V by adding 3D convolutions and temporal attention layers to the U-Net.
- Unsupervised T2V training on video data, bypassing the need for paired text-video datasets.

14.1 Architecture & Method

In figure 78:

- The T2I model doesn't have "fps" input, the decoder generates only a single frame, there are interpolation networks, there are two spatial SR networks and no spatio-temporal SR network.
- The T2V input is a text prompt x and "fps" (frames per second).

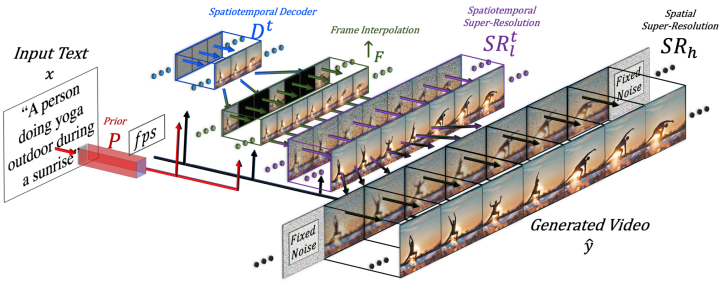


Figure 78: Make-a-Video model architecture overview [82].

- It's not shown in the figure (omitted), but the input text x is converted to **BPE encoding** $\hat{x}$ and given as an input to the prior P network (explained below). However, it's used in equation 17.
- The **CLIP text encoder** C_x (not shown in the figure) is used to translate the text prompt x into text embeddings x_e, which are fed to the prior network P.
- The **image embedding prior** P translates these text embeddings x_e into image embeddings y_e.
- The **spatio-temporal decoder** D^t generates 16 64×64 frames, conditioned on these image embeddings y_e.
- These 16 images are interpolated into higher fps by the **frame interpolation model** $\uparrow_F$, through masked interpolation prediction.
- Then these frames are increased in spatial resolution to 256×256 by the **spatiotemporal super-resolution model** SR_l^t.
- And increased again to resolution 768×768 by **spatial super-resolution model** SR_h.
- The final output is high-spatiotemporal-resolution video $\hat{y}$.

The formal mathematical formulation of the Make-a-Video is:

$$\hat{y}_t = \text{SR}_h \circ \text{SR}_l^t \circ \uparrow_F \circ D^t \circ P \circ (\hat{x}, C_x(x)) \tag{17}$$

14.2 The T2I model

The T2I model in Make-a-Video builds on OpenAI's unCLIP model (DALL-E 2, see section 14.2.1). The T2I model is trained first, and then factorized into a T2V model to extend spatial knowledge to video (explained in section 14.3). After inference (equation 17), images are downsampled to 512×512 using bicubic interpolation for cleaner aesthetics.

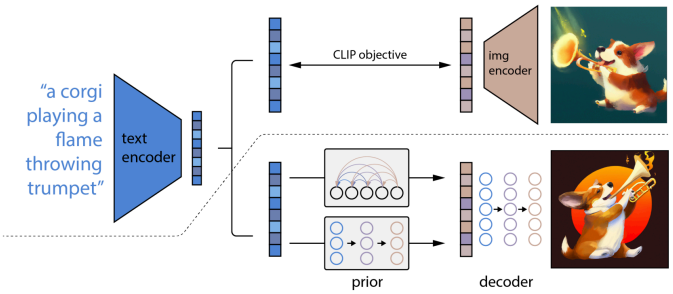


Figure 79: High level overview of unCLIP (DALL-E 2) [66]. *Top*: the contrastive training objective of DALL-E 2; *Bottom*: the inference process [66].

14.2.1 DALL-E 2

DALL-E 2 by OpenAI [66] is a diffusion-based T2I model, which leverages contrastive models like CLIP for image generation. They proposed a two-stage model:

- a prior network $P(z_i|y)$ that generates CLIP image embeddings z_i conditioned on captions
- and a diffusion based decoder $P(x|z_i, y)$ that generates images x conditioned on these image embeddings z_i, and optionally the captions y.

In figure 79 the prior model takes the CLIP text embeddings z_t (the blue vector in the figure) and generates image embeddings z_i. Then the decoder (which acts as inverted CLIP encoder) takes in image embeddings z_i (the brown vector in the figure) and generates the final image:

- **Input**: the training dataset are pairs (x, y) of images x and captions y.
- **Output**: a single generated image x given a caption y.
- **CLIP encoder** is applied to the text x and caption y to get text and image embeddings z_i and z_t respectively.
- **Training objective**: the text embedding z_t should match the CLIP objective to the image embeddings z_i.
- **The prior network** maps the text embeddings z_t to a *distribution* of possible image embedding vectors z_i. In their experiments, they tried two different prior networks:
  - **Autoregressive prior**: the image embedding z_i is converted to a sequence of discrete codes and predicted autoregressively conditioned on the caption y. This network is based on the transformer architecture.
  - **Diffusion based prior** where the latent vector z_i is modeled using Gaussian diffusion model conditioned on caption y. The diffusion model is a decoder-only transformer, instead of a standard U-Net, similar to Diffusion Transformer [58] (appendix G).
- **Decoder network**: given image embeddings z_i, the unCLIP network generates the final image.

- **Classifier-free guidance**: they also used CFG to improve the quality of the generated images and match the captions.
- **Sampling in DALL-E**: first we sample z_i using the prior network, and then we sample x using the decoder (given z_i).
- **Super-resolution**: they also trained two SR diffusion-based networks to upsample the final generated image from 64×64 to 256×256 and finally to 1024×1024.
- **x-prediction**: In the prior network, instead of predicting the noise directly (ϵ-prediction), they chose to predict the unnoised z_i directly (called x-prediction).

14.3 Expanding the T2I model to video domain

The researchers modify the T2I model and transfer the image knowledge to video domain by expanding the 2D conditional network into the temporal dimension.

In short, they support the temporal dimension by adding 3D convolutions temporal attention layers in the U-Net backbone of the T2I diffusion model. The fully-connected layers doesn't need factorization since they are agnostic to structured spatial and temporal information.

14.3.1 Spatiotemporal layers

The spatiotemporal decoder D^t generates 16 RGB frames at 64×64 resolution instead of a single frame. The frame interpolation network $\uparrow_F$ increases the FPS by interpolating these frames (figure 78).

Hallucination: the super-resolution model involves hallucinating information. In order not to have flickering artifacts, the hallucination must be consistent across frames. As a result, SR_l^t operates across spatial and temporal dimensions. In order to consistent detail hallucination across frames, in SR_h they use the same noise initialization for each frame.

In addition, $\text{SR}_\text{l}^\text{t}$ operates across both spatial and temporal dimensions. The spatial layers of SR_l^t were modified for spatiotemporal super-resolution ($\text{SR}_\text{l}^\text{t}$), but SR_h avoids the temporal dimension due to memory and compute constraints further down the pipeline. Consistent detail hallucination in SR_h is achieved by using the same noise initialization for all frames.

14.3.2 Pseudo-3D convolutional layers

Due to the high compute and memory demands of 3D convolutions, the researchers used pseudo-3D convolutional layers, inspired by [15]. This approach decouples spatial and temporal processing via depthwise separable convolution layers, enabling more efficient computation. Pseudo-3D convolutions first apply 2D convolutions on the spatial axes, followed by 1D convolutions on the temporal axis, facilitating information sharing. The formulation is:

$$\text{Conv}_{\text{P3D}}(h) := \text{Conv}_{\text{1D}}(\text{Conv}_{\text{2D}}(h) \circ T) \circ T$$

where $h \in \mathbb{R}^{B \times C \times F \times H \times W}$ is an input tensor, where B, C, F, H, W are the batch, channels, frames, height and width respectively, and $\circ T$ is the transpose operator which swaps between spatial and temporal dimensions.

In figure 80 we can see the pseudo-3D convolution initialization. The temporal 1D convolution layer is initialized as the identity function: it performs no transformation on the input data initially.

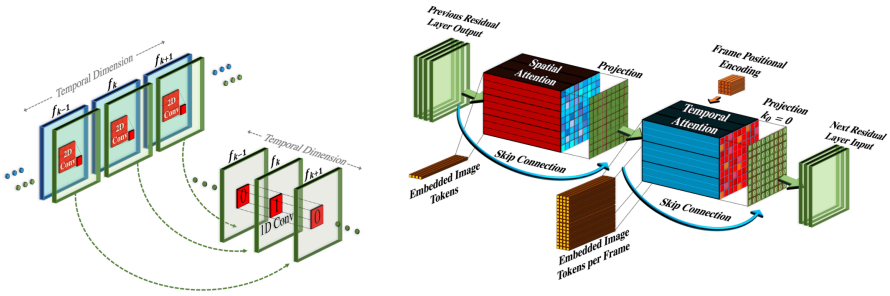


Figure 80: Initialization of pseudo-3D convolutional (left) and temporal attention (right) layers [82].

This way the network relies on the already learned spatial features. The temporal consistency will be learned at a later stage, where the model is trained on video dataset.

14.3.3 Pseudo-3D attention layers

Temporal attention is computationally expensive, so the researchers used stacked pseudo-3D temporal attention layers, inspired by [36] (Video U-Net) and Imagen-Video [35].

Each pre-trained spatial attention layer is followed by a pseudo-3D temporal attention layer. The process involves flattening the spatial dimensions $h' \in \mathbb{R}^{B \times C \times F \times \mathbf{HW}}$ (`unflatten` is the inverse operation) and applying 2D and 1D attention sequentially:

$$\text{ATTN}_{\text{P3D}}(h) = \text{unflatten}(\text{ATTN}_{\text{1D}}(\text{ATTN}_{\text{2D}}(\text{flatten}(h)) \circ T) \circ T)$$

ATTN_{2D} is initialized from the T2I model, and ATTN_{1D} starts as the identity function (figure 80) which allows smooth spatiotemporal initialization.

They also introduce FPS conditioning, inspired by [37], by adding a fps parameter which enables additional augmentation that deals with the limited volume of training videos and provides more control over the generated video at inference time.

14.4 Frame interpolation network

The frame interpolation network $\uparrow_F$ increases number of frames by interpolation or pre/post extrapolation. In addition to RGB channels, they add the binary mask channel (0 or 1) indicating which frames masked during training. This model is conditioned on fps which enable multiple temporal upsample rates at inference. For extrapolation, they can use the same strategy.

14.5 Training

The prior network is trained on text-image pairs and is not fine-tuned for videos.

Initially, the decoder, prior, and two super-resolution (SR) networks are trained on images. Temporal layers are then added, and the models are fine-tuned using unlabeled video data.

To train the T2I model, they used 2.3B subset of Laion-5b [80] of image-text pairs.

To train the T2V model, they used:

- WebVid-10M [3] text-to-video dataset - which was used to train both the decoder, the interpolation network, and the SR_h networks.
- and a 10M subset from HD-VILA-100M [105] - which was used to train SR_h.

14.6 Experiments

They conducted evaluation on UCF-101 [87] and MSR-VTT [104] datasets in zero-shot setting. The researchers also used DrawBench prompts from Imagen for human evaluation.

14.6.1 Quantitative Results

Method	Zero-Shot	Samples/Input	FID (↓)	CLIPSIM (↑)
GODIVA (Wu et al., 2021a)	No	30	–	0.2402
NÜWA (Wu et al., 2021b)	No	–	47.68	0.2439
CogVideo (Hong et al., 2022) (Chinese)	Yes	1	24.78	0.2614
CogVideo (Hong et al., 2022) (English)	Yes	1	23.59	0.2631
Make-A-Video (ours)	Yes	1	**13.17**	**0.3049**

Figure 81: Zero-shot MSR-VTT evaluation of Make-a-Video [82].

Evaluation on MSR-VTT [104] text-video dataset is shown in figure 81. Make-a-Video significantly outperforms all state-of-the-art T2V generation models in zero-shot setting in both FID and CLIP-SIM score.

Method	Pretrain	Class	Resolution	IS (↑)	FVD (↓)
Zero-Shot Setting					
CogVideo (Chinese)	No	Yes	480 × 480	23.55	751.34
CogVideo (English)	No	Yes	480 × 480	25.27	701.59
Make-A-Video (ours)	No	Yes	256 × 256	**33.00**	**367.23**
Finetuning Setting					
TGANv2(Saito et al., 2020)	No	No	128 × 128	26.60 ± 0.47	-
DIGAN(Yu et al., 2022b)	No	No		32.70 ± 0.35	577 ± 22
MoCoGAN-HD(Tian et al., 2021)	No	No	256 × 256	33.95 ± 0.25	700 ± 24
CogVideo (Hong et al., 2022)	Yes	Yes	160 × 160	50.46	626
VDM (Ho et al., 2022)	No	No	64 × 64	57.80 ± 1.3	-
TATS-base(Ge et al., 2022)	No	Yes	128 × 128	79.28 ± 0.38	278 ± 11
Make-A-Video (ours)	Yes	Yes	256 × 256	**82.55**	**81.25**

Figure 82: Zero-shot and fine-tuning evaluation on UCF-101 dataset [82].

Evaluation on UCF-101 is shown in figure 82. Make-a-video also outperforms all the other models in IS, FVD metrics.

In human evaluation (shown in figure 83), they asked humans which video (out of two videos chosen randomly) is higher quality. For faithfulness, they show the text description of the video in addition to the video, and ask which video has better correspondence with the text (text-video alignment).

Comparison	Benchmark	Quality	Faithfulness
Make-A-Video (ours) vs. VDM	VDM prompts (28)	84.38	78.13
Make-A-Video (ours) vs. CogVideo (Chinese)	DrawBench (200)	76.88	73.37
Make-A-Video (ours) vs. CogVideo (English)	DrawBench (200)	74.48	68.75
Make-A-Video (ours) vs. CogVideo (Chinese)	Our Eval. Set (300)	73.44	75.74
Make-A-Video (ours) vs. CogVideo (English)	Our Eval. Set (300)	77.15	71.19

Figure 83: Make-a-Video human evaluation on DrawBench benchmark. The numbers in () indicate amount of videos compared in each evaluation [82].

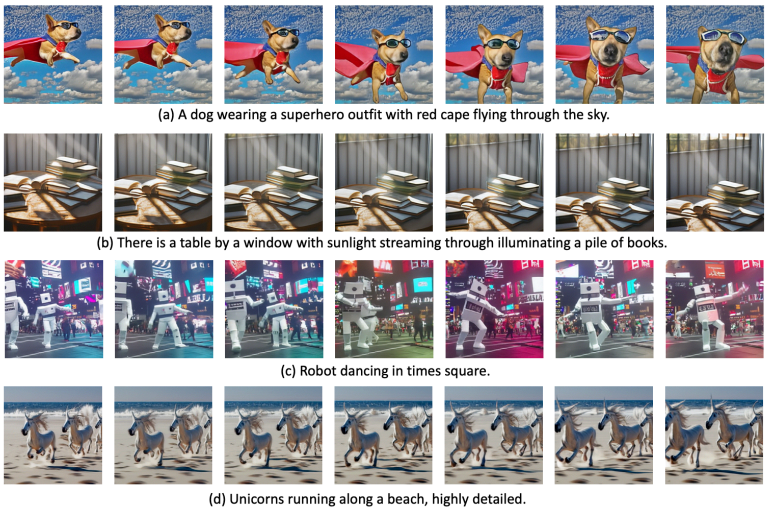


Figure 84: Examples of T2V samples of Make-a-Video [82].

14.6.2 Qualitative Results

The qualitative results of Make-a-Video is remarkable. We see some samples in figures 84 and 85.

In figure 85 (c) we see the interpolation network works better than FILM [69] by Google, which is a model specifically learned to interpolate between frames.

15 Conclusion

This work explores recent advancements in deep learning techniques for image and video generation, focusing on the transition from the mature domain of image synthesis to the emerging challenges of video synthesis.

We examined foundational models, including VAEs, GANs, and DMs based models. We explored the state-of-the-art models like VQ-GAN, LDM, and Imagen for image generation and Video-LDM, Imagen-Video, and Make-a-Video for video generation.

Here is a brief summary of the models discussed:

- **VQ-VAEs**: Probabilistic models employing latent variables for structured image reconstructions, allowing smooth interpolation in the latent space. VQ-VAE enhances VAEs by in-

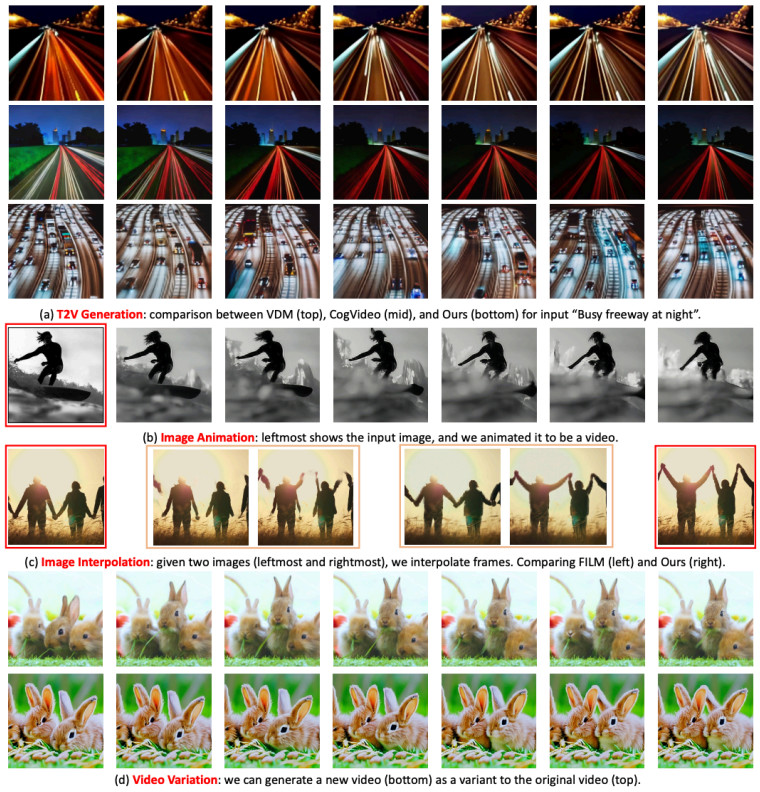


Figure 85: Various qualitative results comparisons and applications [82].

troducing vector quantization, discretizing the latent space to facilitate more efficient prior learning and sample generation.

- **GANs**: Adversarial models that learn the data distribution by training a generator to produce realistic samples and a discriminator to distinguish between real and generated samples. However, GANs suffer from instability during training due to adversarial loss, leading to mode collapse and convergence issues. As a result, their usage has declined with the rise of diffusion-based methods.
- **Stable Diffusion**: Probabilistic model that learns the data distribution by iteratively applying noise to the data and then learns to denoise iteratively. Stable Diffusion is often the basis of most image and video generation models because of it is stability and high-fidelity outputs.
- **Imagen**: Imagen leverages transformers for text-to-image (T2I) generation, combining their strengths with cascaded diffusion models and super-resolution techniques to achieve state-of-the-art performance. They have shown that the more parameters in the transformer, the better the FID and CLIP evaluation scores.

We then shift focus to video synthesis, which builds upon image based techniques to address temporal challenges:

- **VideoGPT**: A transformer-based model that generates videos by predicting future frames conditioned on past frames. It uses VQ-VAE to operate in the latent space instead of the pixel space, which is more compute-efficient.
- **Video-LDM**: Takes pre-trained T2I LDM model to video domain by freezing the spatial layers and adding temporal attention and 3D convolution layers in order to fine-tune to video data. It operates in the latent space and incorporates a GAN discriminator to enhance the temporal coherence of generated video.
- **Imagen-Video**: a T2V model based on the previous work of Imagen, it extends Imagen to video generation, employing a cascaded diffusion framework with seven diffusion-based models to enhance spatial and temporal resolution.
- **Make-a-Video**: A model that is based on the work of DALL-E 2, which employs pseudo-3D convolutions and pseudo-temporal attention to balance computational efficiency with generative quality, addressing the high cost of full temporal attention and 3D convolutions.

A big challenge in T2V models is the heavy training cost associated with T2V models, with some tasks requiring the use of hundreds of GPUs. Despite many efforts to reduce the training cost, both the magnitude of dataset and temporal complexity remains a critical concern. More efficient compression of video representations, exploration of effective spatiotemporal blocks and acceleration of training and inference times are essential for the future of long video synthesis.

Autoregressive methods for generating long videos suffer from error accumulation [56], resulting in poorer quality in later frames. Moreover, most video generation models currently can only produce videos shorter than 10 seconds.

In summary, the progression from image to video generation models illustrates the evolving capabilities of generative AI. While image synthesis models have reached a level of maturity with highly realistic outputs, video synthesis remains a frontier, requiring innovative approaches to address temporal coherence and computational challenges.

References

[1] *AE v.s. VAE figure source*. URL: https://vitalflux.com/autoencoder-vs-variational-autoencoder-vae-difference/.

[2] Jimmy Lei Ba. "Layer normalization." In: *arXiv preprint arXiv:1607.06450* (2016).

[3] Max Bain et al. "Frozen in time: A joint video and image encoder for end-to-end retrieval." In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 1728–1738.

[4] Dana H Ballard. "Modular learning in neural networks." In: *Proceedings of the sixth National conference on Artificial intelligence-Volume 1*. 1987, pp. 279–284.

[5] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. "Estimating or propagating gradients through stochastic neurons for conditional computation." In: *arXiv preprint arXiv:1308.3432* (2013).

[6] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult." In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[7] Bilel Benjdira et al. "Data-efficient domain adaptation for semantic segmentation of aerial imagery using generative adversarial networks." In: *Applied Sciences* 10.3 (2020), p. 1092.

[8] Andreas Blattmann et al. "Align your latents: High-resolution video synthesis with latent diffusion models." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 22563–22575.

[9] Andreas Blattmann et al. "Stable video diffusion: Scaling latent video diffusion models to large datasets." In: *arXiv preprint arXiv:2311.15127* (2023).

[10] Andrew Brock. "Large Scale GAN Training for High Fidelity Natural Image Synthesis." In: *arXiv preprint arXiv:1809.11096* (2018).

[11] Tim Brooks et al. "Generating long videos of dynamic scenes." In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 31769–31781.

[12] Tom B Brown. "Language models are few-shot learners." In: *arXiv preprint arXiv:2005.14165* (2020).

[13] Mark Chen et al. "Generative pretraining from pixels." In: *International conference on machine learning*. PMLR. 2020, pp. 1691–1703.

[14] Xi Chen et al. "Infogan: Interpretable representation learning by information maximizing generative adversarial nets." In: *Advances in neural information processing systems* 29 (2016).

[15] Francois Chollet. "Xception: Deep learning with depthwise separable convolutions." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1251–1258.

[16] *Common Crawl website*. URL: https://commoncrawl.org/.

[17] Katherine Crowson et al. "Vqgan-clip: Open domain image generation and editing with natural language guidance." In: *European Conference on Computer Vision*. Springer. 2022, pp. 88–105.

[18] Jacob Devlin. "Bert: Pre-training of deep bidirectional transformers for language understanding." In: *arXiv preprint arXiv:1810.04805* (2018).

[19] Prafulla Dhariwal and Alexander Nichol. "Diffusion models beat gans on image synthesis." In: *Advances in neural information processing systems* 34 (2021), pp. 8780–8794.

[20] Alexey Dosovitskiy et al. "An image is worth 16x16 words: Transformers for image recognition at scale." In: *arXiv preprint arXiv:2010.11929* (2020).

[21] Patrick Esser, Robin Rombach, and Bjorn Ommer. "Taming transformers for high-resolution image synthesis." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 12873–12883.

[22] Gereon Fox et al. "Stylevideogan: A temporal generative model using a pretrained stylegan." In: *arXiv preprint arXiv:2107.07224* (2021).

[23] *GAN mode-collapse image source*. URL: https://pub.towardsai.net/gan-mode-collapse-explanation-fa5f9124ee73.

[24] Ian Goodfellow et al. "Generative adversarial nets." In: *Advances in neural information processing systems* 27 (2014).

[25] Alex Graves. "Generating sequences with recurrent neural networks." In: *arXiv preprint arXiv:1308.0850* (2013).

[26] Jiaxi Gu et al. "Reuse and diffuse: Iterative denoising for text-to-video generation." In: *arXiv preprint arXiv:2309.03549* (2023).

[27] Kaiming He et al. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[28] Joel Hestness et al. "Deep learning scaling is predictable, empirically." In: *arXiv preprint arXiv:1712.00409* (2017).

[29] Martin Heusel et al. "Gans trained by a two time-scale update rule converge to a local nash equilibrium." In: *Advances in neural information processing systems* 30 (2017).

[30] Geoffrey E Hinton and Ruslan R Salakhutdinov. "Reducing the dimensionality of data with neural networks." In: *science* 313.5786 (2006), pp. 504–507.

[31] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models." In: *Advances in neural information processing systems* 33 (2020), pp. 6840–6851.

[32] Jonathan Ho and Tim Salimans. "Classifier-free diffusion guidance." In: *arXiv preprint arXiv:2207.12598* (2022).

[33] Jonathan Ho et al. "Axial attention in multidimensional transformers." In: *arXiv preprint arXiv:1912.12180* (2019).

[34] Jonathan Ho et al. "Cascaded diffusion models for high fidelity image generation." In: *Journal of Machine Learning Research* 23.47 (2022), pp. 1–33.

[35] Jonathan Ho et al. "Imagen video: High definition video generation with diffusion models." In: *arXiv preprint arXiv:2210.02303* (2022).

[36] Jonathan Ho et al. "Video diffusion models." In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 8633–8646.

[37] Wenyi Hong et al. "Cogvideo: Large-scale pretraining for text-to-video generation via transformers." In: *arXiv preprint arXiv:2205.15868* (2022).

[38] Sergey Ioffe. “Batch normalization: Accelerating deep network training by reducing internal covariate shift.” In: *arXiv preprint arXiv:1502.03167* (2015).

[39] Phillip Isola et al. “Image-to-image translation with conditional adversarial networks.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1125–1134.

[40] Rafal Jozefowicz et al. “Exploring the limits of language modeling.” In: *arXiv preprint arXiv:1602.02410* (2016).

[41] Tero Karras, Samuli Laine, and Timo Aila. “A style-based generator architecture for generative adversarial networks.” In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 4401–4410.

[42] Tero Karras et al. “Alias-free generative adversarial networks.” In: *Advances in neural information processing systems* 34 (2021), pp. 852–863.

[43] Diederik Kingma et al. “Variational diffusion models.” In: *Advances in neural information processing systems* 34 (2021), pp. 21696–21707.

[44] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes.” In: *arXiv preprint arXiv:1312.6114* (2013).

[45] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks.” In: *Advances in neural information processing systems* 25 (2012).

[46] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning.” In: *nature* 521.7553 (2015), pp. 436–444.

[47] Chengxuan Li et al. “A survey on long video generation: Challenges, methods, and prospects.” In: *arXiv preprint arXiv:2403.16407* (2024).

[48] Tsung-Yi Lin et al. “Microsoft coco: Common objects in context.” In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer. 2014, pp. 740–755.

[49] Chenlin Meng et al. “On distillation of guided diffusion models.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 14297–14306.

[50] *Midjourney website*. URL: https://www.midjourney.com/home.

[51] Tomas Mikolov. “Efficient estimation of word representations in vector space.” In: *arXiv preprint arXiv:1301.3781* (2013).

[52] Mehdi Mirza and Simon Osindero. “Conditional generative adversarial nets.” In: *arXiv preprint arXiv:1411.1784* (2014).

[53] Charlie Nash et al. “Transframer: Arbitrary frame prediction with generative models.” In: *arXiv preprint arXiv:2203.09494* (2022).

[54] Alex Nichol et al. “Glide: Towards photorealistic image generation and editing with text-guided diffusion models.” In: *arXiv preprint arXiv:2112.10741* (2021).

[55] Alexander Quinn Nichol and Prafulla Dhariwal. “Improved denoising diffusion probabilistic models.” In: *International conference on machine learning*. PMLR. 2021, pp. 8162–8171.

[56] Yichen Ouyang, Hao Zhao, Gaoang Wang, et al. “FlexiFilm: Long Video Generation with Flexible Conditions.” In: *arXiv preprint arXiv:2404.18620* (2024).

[57] Gaurav Parmar, Richard Zhang, and Jun-Yan Zhu. “On aliased resizing and surprising subtleties in gan evaluation.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 11410–11420.

[58] William Peebles and Saining Xie. “Scalable diffusion models with transformers.” In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 4195–4205.

[59] Simon JD Prince. *Understanding deep learning*. MIT press, 2023.

[60] Alec Radford. “Improving language understanding by generative pre-training.” In: (2018).

[61] Alec Radford et al. “Language models are unsupervised multitask learners.” In: *OpenAI blog* 1.8 (2019), p. 9.

[62] Alec Radford et al. “Language models are unsupervised multitask learners.” In: *OpenAI blog* 1.8 (2019), p. 9.

[63] Alec Radford et al. “Learning transferable visual models from natural language supervision.” In: *International conference on machine learning*. PMLR. 2021, pp. 8748–8763.

[64] Colin Raffel et al. “Exploring the limits of transfer learning with a unified text-to-text transformer.” In: *Journal of machine learning research* 21.140 (2020), pp. 1–67.

[65] Prajit Ramachandran, Barret Zoph, and Quoc V Le. “Swish: A Self-Gated Activation Function.” In: *arXiv preprint arXiv:1710.05941* (2017).

[66] Aditya Ramesh et al. “Hierarchical text-conditional image generation with clip latents.” In: *arXiv preprint arXiv:2204.06125* 1.2 (2022), p. 3.

[67] Aditya Ramesh et al. “Zero-shot text-to-image generation.” In: *International conference on machine learning*. Pmlr. 2021, pp. 8821–8831.

[68] Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. “Generating diverse high-fidelity images with vq-vae-2.” In: *Advances in neural information processing systems* 32 (2019).

[69] Fitsum Reda et al. “Film: Frame interpolation for large motion.” In: *European Conference on Computer Vision*. Springer. 2022, pp. 250–266.

[70] Alex Rogozhnikov. “Einops: Clear and reliable tensor manipulations with einstein-like notation.” In: *International Conference on Learning Representations*. 2021.

[71] Robin Rombach et al. “High-resolution image synthesis with latent diffusion models.” In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 10684–10695.

[72] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “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*. Springer. 2015, pp. 234–241.

[73] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning internal representations by error propagation, parallel distributed processing, explorations in the microstructure of cognition, ed. de rumelhart and j. mcclelland. vol. 1. 1986.” In: *Biometrika* 71.599-607 (1986), p. 6.

[74] Chitwan Saharia et al. “Image super-resolution via iterative refinement.” In: *IEEE transactions on pattern analysis and machine intelligence* 45.4 (2022), pp. 4713–4726.

[75] Chitwan Saharia et al. “Photorealistic text-to-image diffusion models with deep language understanding.” In: *Advances in neural information processing systems* 35 (2022), pp. 36479–36494.

[76] Tim Salimans and Jonathan Ho. “Progressive distillation for fast sampling of diffusion models.” In: *arXiv preprint arXiv:2202.00512* (2022).

[77] Tim Salimans et al. “Improved techniques for training gans.” In: *Advances in neural information processing systems* 29 (2016).

[78] Jürgen Schmidhuber. “Deep learning in neural networks: An overview.” In: *Neural networks* 61 (2015), pp. 85–117.

[79] Christoph Schuhmann et al. “Laion-400m: Open dataset of clip-filtered 400 million image-text pairs.” In: *arXiv preprint arXiv:2111.02114* (2021).

[80] Christoph Schuhmann et al. “Laion-5b: An open large-scale dataset for training next generation image-text models.” In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 25278–25294.

[81] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition.” In: *arXiv preprint arXiv:1409.1556* (2014).

[82] Uriel Singer et al. “Make-a-video: Text-to-video generation without text-video data.” In: *arXiv preprint arXiv:2209.14792* (2022).

[83] Jascha Sohl-Dickstein et al. “Deep unsupervised learning using nonequilibrium thermodynamics.” In: *International conference on machine learning*. PMLR. 2015, pp. 2256–2265.

[84] Jiaming Song, Chenlin Meng, and Stefano Ermon. “Denoising diffusion implicit models.” In: *arXiv preprint arXiv:2010.02502* (2020).

[85] Yang Song et al. “Score-based generative modeling through stochastic differential equations.” In: *arXiv preprint arXiv:2011.13456* (2020).

[86] Yang Song et al. “Score-based generative modeling through stochastic differential equations.” In: *arXiv preprint arXiv:2011.13456* (2020).

[87] K Soomro. “UCF101: A dataset of 101 human actions classes from videos in the wild.” In: *arXiv preprint arXiv:1212.0402* (2012).

[88] *Sora by OpenAI website*. 2024. URL: https://openai.com/index/sora.

[89] Rui Sun et al. “From Sora What We Can See: A Survey of Text-to-Video Generation.” In: *arXiv preprint arXiv:2405.10674* (2024).

[90] Christian Szegedy et al. “Rethinking the inception architecture for computer vision.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[91] Yu Tian et al. “A good image generator is what you need for high-resolution video synthesis.” In: *arXiv preprint arXiv:2104.15069* (2021).

[92] Du Tran et al. “A closer look at spatiotemporal convolutions for action recognition.” In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2018, pp. 6450–6459.

[93] Du Tran et al. “Learning spatiotemporal features with 3d convolutional networks.” In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 4489–4497.

[94] Thomas Unterthiner et al. “FVD: A new metric for video generation.” In: (2019).

[95] Aaron Van Den Oord, Oriol Vinyals, et al. “Neural discrete representation learning.” In: *Advances in neural information processing systems* 30 (2017).

[96] Aaron Van den Oord et al. “Conditional image generation with pixelcnn decoders.” In: *Advances in neural information processing systems* 29 (2016).

[97] Ashish Vaswani et al. “Attention is all you need.” In: *Advances in neural information processing systems* 30 (2017).

[98] *Vector quantization visualization, website source*. URL: https://speechprocessingbook.aalto.fi/Modelling/Vector_quantization_VQ.html.

[99] Subhashini Venugopalan et al. “Sequence to sequence-video to text.” In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 4534–4542.

[100] *Why the round function is not differentiable*. URL: https://ai.stackexchange.com/questions/26770/in-vq-vae-code-what-does-this-line-of-code-signify.

[101] Chenfei Wu et al. “Nuwa-infinity: Autoregressive over autoregressive generation for infinite visual synthesis.” In: *arXiv preprint arXiv:2207.09814* (2022).

[102] Yuxin Wu and Kaiming He. “Group normalization.” In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 3–19.

[103] Jiaqi Xu et al. “EasyAnimate: A High-Performance Long Video Generation Method based on Transformer Architecture.” In: *arXiv preprint arXiv:2405.18991* (2024).

[104] Jun Xu et al. “Msr-vtt: A large video description dataset for bridging video and language.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 5288–5296.

[105] Hongwei Xue et al. “Advancing high-resolution video-language representation with large-scale video transcriptions.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 5036–5045.

[106] Wilson Yan et al. “Videogpt: Video generation using vq-vae and transformers.” In: *arXiv preprint arXiv:2104.10157* (2021).

[107] Shengming Yin et al. “Nuwa-xl: Diffusion over diffusion for extremely long video generation.” In: *arXiv preprint arXiv:2303.12346* (2023).

[108] Sihyun Yu et al. “Video probabilistic diffusion models in projected latent space.” In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2023, pp. 18456–18466.

[109] Yan Zeng et al. “Make pixels dance: High-dynamic video generation.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 8850–8860.

[110] Richard Zhang et al. “The unreasonable effectiveness of deep features as a perceptual metric.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 586–595.

[111] Daquan Zhou et al. “Magicvideo: Efficient video generation with latent diffusion models.” In: *arXiv preprint arXiv:2211.11018* (2022).

[112] Pengyuan Zhou et al. “A survey on generative ai and llm for video generation, understanding, and streaming.” In: *arXiv preprint arXiv:2404.16038* (2024).

[113] Jun-Yan Zhu et al. “Unpaired image-to-image translation using cycle-consistent adversarial networks.” In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2223–2232.

[114] Vicky Zilvan et al. “Convolutional variational autoencoder-based feature learning for automatic tea clone recognition.” In: *Journal of King Saud University-Computer and Information Sciences* 34.6 (2022), pp. 3332–3342.

A Appendix

A Likelihood function

Likelihood function in the realm of generative models, is often used to capture the underlying data distribution (generate data points with similar likelihood to the training data distribution).

The formal notion of likelihood function is: $L(x|\theta)$, which reflects the probability of a specific data point (x) being generated by the model with its current parameters (θ). We want to maximize this term: $\arg\max_{\theta} L(x|\theta)$. In many cases it's computationally more convenient to maximize the **log-likelihood** function instead, as the logarithm is a monotonic function (always increasing or decreasing, and therefor the log of the function is also monotonic):

$$\hat{\theta}_{MLE} = \arg\max_{\theta} \log L(\mathbf{x}|\theta)$$

Since directly calculating the likelihood is often intractable, methods like **maximum likelihood estimation (MLE)** optimize θ indirectly. To address intractability, approaches such as **Evidence Lower Bound (ELBO)** are used as alternative loss functions. Additionally, adversarial training is widely employed in GAN-based models.

B Activation functions

Sigmoid

Sigmoid (figure 86a) is an activation function used in many neural networks. It is one of the most common activation functions. It normalizes the output in the range [0,1] and is usually used in binary classification or logistic regression tasks. In binary classification task it can be used as the last layer of the neural network, which would output a single probability output. It is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

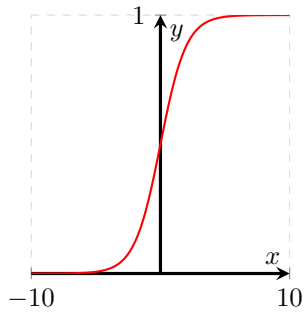
ReLU

ReLU (Rectified Linear Unit) (figure 86b) is an activation function used in many neural networks. It is defined as:

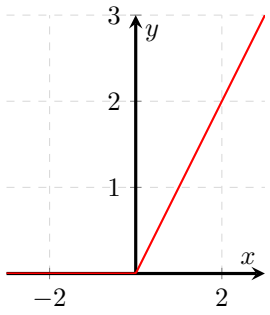
$$\text{ReLU}(x) = \max(0, x)$$

Softmax

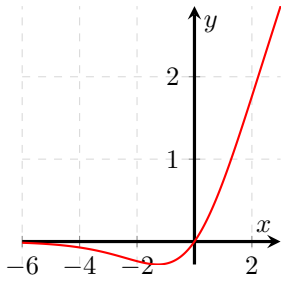
Softmax is an activation function, usually used in the output layer of a neural network. It is used to compress multiple input values into a range between [0,1] similarly to sigmoid, but softmax can be thought as a probability distribution, where multiple values are mapped to a probability. Softmax is often used in multi-class classification. It is defined as:



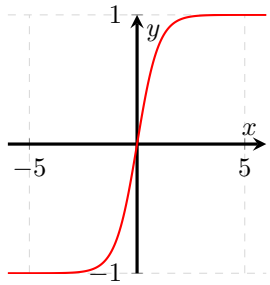
(a) Sigmoid activation function.



(b) ReLU activation function.



(c) Swish activation function.



(d) Tanh activation function.

Figure 86: Common activation functions used in deep learning.

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

where x_i is the i-th element of the input vector x and n is the number of elements in the input vector x.

Tanh

Tanh (figure 86d) is often used in RNNs (Recurrent Neural Networks) and LSTMs (Long Short-Term Memory) gates. Although these models are out of scope in this work, it is still worth mentioning as this activation function is used in many models, such as GANs. The output range is [-1, 1] and it is zero centered. It is defined as:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Swish

Swish (figure 86c) is an activation function used in the Imagen `ResNetBlock`. It was first introduced in a 2017 paper [65] by Google Brain team. It is defined as:

$$\text{Swish}(x) = x \cdot \sigma(x) \tag{18}$$

where $\sigma(x)$ is the sigmoid activation function.

Swish is similar to ReLU activation function, but it allows smoother gradients during training, and often it outperforms ReLU in some deep learning tasks.

C Common neural network blocks

C.1 Multi-layer perception (MLP)

Multi-layer perception (MLP) is a basic feed forward neural network architecture in which the input is passed through multiple layers of linear transformations (fully connected layers) and non-linear activation functions.

In short, we can think of MLP as a simple fully connected layers (sometimes called 'dense layer').

C.2 ResBlock

Residual blocks (`ResBlock`) are skip-connection blocks that were introduced in the ResNet (Residual Network) paper [27]. They are used to mitigate the exploding/vanishing gradient problem in very deep neural networks. A residual block allows the network to skip layers by adding identity shortcut connection (the input is added directly to the output of a few stacked layers). The main idea is that instead of learning the full mappings from input to output, the residual block only needs to learn a residual mapping (the difference between the input and output). Mathematically, residual mappings can be written as:

$$y = \mathcal{F}(x) + x$$

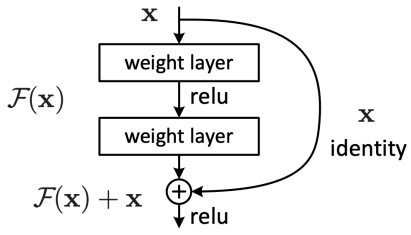


Figure 87: `ResBlock` architecture, from the ResNet paper [27].

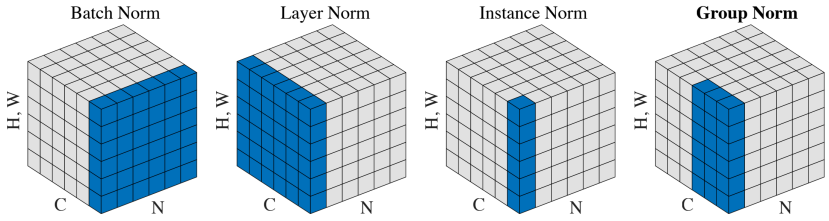


Figure 88: Visual comparison on mode of operations of `BatchNorm`, `LayerNorm`, `InstanceNorm`, and `GroupNorm` [102].

where x is the input to the block, $\mathcal{F}$ is the residual function, and y is the output of the block.

If a layer doesn't need to transform the input, the skip connection allows the network to learn an identity mapping, making it easier to optimize the model. Easier flow of gradients also mitigate the problem of vanishing/exploding gradients, where the gradients become too small or too large as they are backpropagated through the network, which is especially prominent in very deep networks.

C.3 Normalization layers

These layers ensure that the data is normalized in a way that the model can learn better.

For example, we can think that the optimization problem is on a distribution that looks like very long ellipse, it would be hard for the model to reach minima. But if we normalize the data such that the distribution will fit a circle, then the optimization process will be easier. Normalization layers are used to change the shape of the distribution, scale it, and shift it to make the optimization process easier.

We use mean μ and variance σ in the normalization layers:

$$\mu = \frac{1}{H}\sum_{i=1}^{H} x_i$$

$$\sigma = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(x_i - \mu)^2}$$

where H is the number of elements in the input x.

LayerNorm

`LayerNorm`, introduced in a 2016 paper [2], is a normalization block that normalizes the input across the feature channels rather than across the batch (as in `BatchNorm` block). This layer scales and shifts the input distribution. See figure 88 for a visual comparison. In addition, because `LayerNorm` doesn't rely on batch size, it can be applied even in batch size of 1.

It's formally defined as:

$$\text{LayerNorm}(x) = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \cdot \gamma + \beta$$

where $\mu = \mathbb{E}(x)$ is the mean of the input, $\sigma = \text{Var}(x)$ is the variance of the input, ϵ is a small constant to avoid division by zero, and γ, β are learnable parameters. The γ and β parameters scale and shift the normalized input.

BatchNorm

`BatchNorm` was introduced in a 2015 paper [38][§§§].

Let μ_B be the mean of the mini-batch, and σ_B^2 be the variance of the mini-batch, and m be the number of samples in the mini-batch. The operation of `BatchNorm` is defined as:

$$\text{BatchNorm}(x) = \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \cdot \gamma + \beta$$

where γ, β are learnable parameters, and ϵ is a small constant to avoid division by zero. The γ and β parameters scale and shift the normalized input.

GroupNorm

`GroupNorm` was introduced in a 2018 paper [102]. `GroupNorm` strikes a balance between `LayerNorm` and `BatchNorm`, where it normalizes the input across the feature channel but not all of it, rather it normalizes groups of input features, hence the name.

The operation of `GroupNorm` is defined as:

$$\text{GroupNorm}(x) = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \cdot \gamma + \beta$$

where $\mu = \mathbb{E}(x)$ is the mean of the input, $\sigma = \text{Var}(x)$ is the variance of the input, ϵ is a small constant to avoid division by zero, and γ, β are learnable parameters. The γ and β parameters scale and shift the normalized input.

C.4 3D Convolutions

Figure 89 illustrates 2D and 3D convolutions.

3D convolution extends 2D convolution by adding a depth dimension, useful for temporal data like videos. The kernel moves through all three dimensions: width, height, and depth (often known as the temporal dimension for video, across frames).

[§§§] A great YouTube video explaining why batch normalization speeds up the training process (by not overshooting in the learning rate) is available at this link.

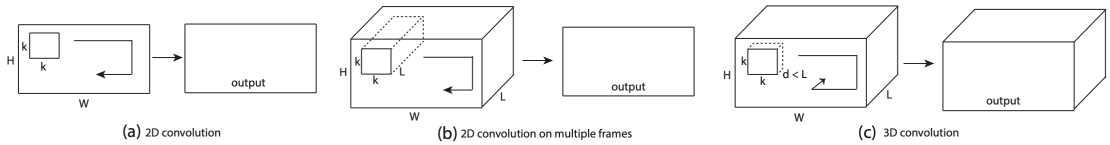


Figure 89: 2D and 3D convolution operations. (a) Applying 2D convolution on image results in an image (2D tensor). (b) Applying 2D convolution on video (multiple frames) also results in an 2D vector. (c) Applying 3D convolution on video results in a 3D tensor with a temporal dimension. Figure from a 2014 paper [93].

Given an input of size $H \times W \times D$ and a kernel of size $k_h \times k_w \times k_d$, the 3D convolution operation is defined as:

$$y_{i,j,k} = \sum_{l=0}^{k_d} \sum_{m=0}^{k_h} \sum_{n=0}^{k_w} x_{i+m,j+n,k+l} \cdot w_{m,n,l}$$

Like 2D convolution, 3D convolution can use stride, padding, and dilation. Stride refers to the kernel's movement steps, padding adds zeros to the input to preserve output size, and dilation controls the spacing between kernel elements.

For both 2D and 3D convolutions, the number of input and output channels determines the number of filters applied.

D Attention mechanism

Attention mechanisms, introduced in the Transformer paper [97], are pivotal in modern deep learning. This section explores the core components of transformers and their role in image and video synthesis models.

D.1 Self-attention

Self-attention allows us to model the weight of different parts of an input sequence. It gives indication on how much focus (**attention score**) to give to different elements in the sequence, and considers relationships between then. In self-attention, the same set of data elements provides the queries, keys and values, unlike cross-attention where we have two different sets of data elements.

Input: A vector $X \in \mathbb{R}^{n \times d}$.

Output: Similarity score between the elements in the sequence (equation 19).

The Q, K, V matrices: The Q, K, V matrices are learnable matrices that are used to calculate the attention scores. We can think of **Q**uery, **K**ey, **V**alue as text search in Google. When we *query* Google, we get *keys* as a result (list of pages), which is not what we are looking for. When we click on a page, then we get the *value*. Therefor Google has to find similarity (attention scores) between query and the keys.

Linear projection: To get the Q, K, V matrices we apply linear projections on the input sequence X:

$$Q = X \cdot W_Q$$
$$K = X \cdot W_K$$
$$V = X \cdot W_V$$

where $W_Q, W_K, W_V \in \mathbb{R}^{d \times d}$ are learnable weight matrices (they are assigned random values initially and learned during training).

Cosine similarity: We can think of similarity in linear algebra as cosine similarity between vectors A, B:

$$\text{CosSim}(A, B) = \frac{A \cdot B}{||A|| \cdot ||B||}$$

Finally, the self-attention is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{19}$$

where d_k is the dimension of the key vector, and dividing by $\sqrt{d_k}$ normalizes the large magnitude of the dot product - which stabilizes the gradients and the training.

We then apply softmax function (see appendix B) to the attention weights, which converts the vector to a probability distribution between 0 and 1, which gives probability to tokens in the sequence.

We then multiply (dot product) the attention weights with the value matrix to get the output. K is transposed to match the dimensions of Q (so we can multiply them).

Listing 3: Full implementation of a single self-attention block.

```
class SelfAttention(nn.Module):
"""
embed_dim (int): The dimensionality of the input embeddings.
attention_dim (int): The dimensionality of the attention space / attention
    vectors (Q, K, V).
"""
def __init__(self, embed_dim, attention_dim):
    super(SelfAttention, self).__init__()
    # Linear projections for Q, K, V
    self.W_Q = nn.Linear(embed_dim, attention_dim)
    self.W_K = nn.Linear(embed_dim, attention_dim)
    self.W_V = nn.Linear(embed_dim, attention_dim)
    self.scale = torch.sqrt(torch.tensor(attention_dim, dtype=torch.float32))

def forward(self, X):
    Q = self.W_Q(X)
    K = self.W_K(X)
    V = self.W_V(X)

    attention_scores = torch.matmul(Q, K.transpose(-2, -1)) / self.scale
    attention_weights = F.softmax(attention_scores, dim=-1)
```

```
    attention_output = torch.matmul(attention_weights, V)

    return attention_output
```

D.2 Multi-head attention

Multi-head attention is a concatenation of multiple self-attention heads, each with its own set of weights W_i^Q, W_i^K, W_i^V. Each attention head can learn different relationships between the same elements in the same sequence. After concatenation, we apply dot product with the output weights matrix W^O:

The input matrices (Q, K, V) derive from the **same input sequence**. In comparison, in cross-attention, we relate two different sequences. Cross-attention is particularly useful in tasks that requires alignment between different sequences, like in translation tasks, whereas in self-attention and multi-head attention we are looking for relationships within the same sequence, like in sentiment analysis.

Multi-head attention is defined as concatenation of different attention heads:

$$\begin{aligned} \text{head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \\ \text{MultiHead}(Q,K,V) &= \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \end{aligned} \tag{20}$$

D.3 Cross-Attention

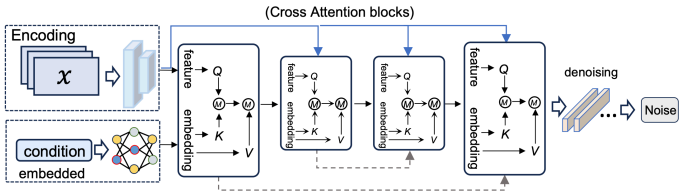


Figure 90: Cross-attention in Stable Diffusion. The K, V matrices are derived from conditional embeddings, while the Q matrix is projected from the image/noise encodings [89].

Cross-attention aligns **two different sequences** by computing attention weights from a query sequence (Q) and applying them to key-value pairs (K, V). This mechanism is particularly effective in conditional generation tasks, such as text-to-image (T2I) and text-to-video (T2V), where text prompts guide the generation process.

In Stable Diffusion (Section 8), cross-attention is employed to condition the latent image encodings on text embeddings. Specifically, the input sequence X (e.g. images) is projected onto Q, while the conditional sequence Y (e.g., text) is projected onto K and V, using:

$$\begin{aligned} Q &= X \cdot W_Q \\ K &= Y \cdot W_K \\ V &= Y \cdot W_V \end{aligned}$$

where $W_Q, W_K, W_V \in \mathbb{R}^{d \times d}$ are learnable projection matrices.

Multi-head cross-attention extends this by using multiple attention heads, each operating independently and concatenated together. Listing 4 shows a simplified PyTorch implementation of cross-attention.

Listing 4: PyTorch implementation of cross-attention. The '@' operator represents the dot product.

```
1  class CrossAttention(nn.Module):
2      def __init__(self, n_heads, d_embed, d_cross,
3                   in_proj_bias=True, out_proj_bias=True):
4          super().__init__()
5          self.q_proj = nn.Linear(d_embed, d_embed, bias=in_proj_bias)
6          self.k_proj = nn.Linear(d_cross, d_embed, bias=in_proj_bias)
7          self.v_proj = nn.Linear(d_cross, d_embed, bias=in_proj_bias)
8          self.out_proj = nn.Linear(d_embed, d_embed, bias=out_proj_bias)
9          self.n_heads = n_heads
10         self.d_head = d_embed // n_heads
11
12     def forward(self, x, y):
13         # Projection and attention computation
14         q = self.q_proj(x)
15         k = self.k_proj(y)
16         v = self.v_proj(y)
17
18         weight = q @ k.transpose(-1, -2) / math.sqrt(self.d_head)
19         weight = F.softmax(weight, dim=-1)
20         output = weight @ v
21
22         # Reshape and project output
23         output = output.transpose(1, 2).contiguous()
24         output = output.view_as(x)
25         return self.out_proj(output)
```

The implementation is adapted from a re-implementation of Stable Diffusion. Note that the official implementation contains a more complex version of cross-attention.

Notice lines 14, 15, 16 in listing 4: the input X is projected onto Q only, and Y is projected onto both K and V.

D.4 Axial attention

Axial attention [33], used in VideoGPT [106], addresses the quadratic complexity of self-attention ($O(n^2)$) by attending to one axis at a time, reducing the complexity to $O(n\sqrt{n})$. Instead of processing the full input sequence, it alternately attends to masked rows and columns, saving computations while preserving global spatial context.

This axial attention masking can be seen in figure 91. The order of pixels that are from top-left to bottom-right and as an autoregressive pixel prediction model, which uses axial attention, will be predicting the pixel $p_{i,j}$ and not $p_{i+1,j}$ or $p_{i,j+1}$, which depends only on previous pixels.

Although axial transformer is not attending to every pixel, it still captures spatial global context more efficiently.

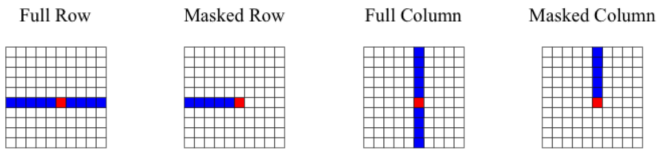


Figure 91: Types of axial attention layers which are the building blocks of axial transformer.

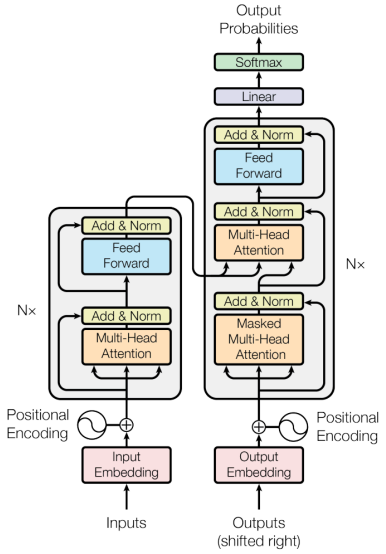


Figure 92: Transformer architecture. The encoder (left) processes the input sequence, and the decoder (right) generates the output sequence [97].

E Transformers

The transformer [97], built on self-attention (appendix D), is widely used in NLP tasks like translation, text generation, and classification. It consists of an encoder for input processing and a decoder for output generation, trained via maximum likelihood estimation (appendix A).

It is versatility in sequence-to-sequence tasks has expanded its applications to image and video synthesis models.

E.1 Architecture

The architecture overview is shown in figure 92. The encoder processes the input sequence and breaks it down to meaningful representations, and the decoder takes these representations and generates output sequence in autoregressive fashion.

Encoder

The encoder processes a sequence of tokens, converted from text by a tokenizer (e.g., BERT, T5). Tokens are mapped to fixed-size vector embeddings through an embedding layer.

Positional embeddings: Since the transformer has no inherent understanding of sequence or order, positional encodings are added to each embedding to provide information about the position of each token in the sequence. Positional embeddings are added to token embeddings (figure 92). The positional embeddings are computed as sinusoidal embeddings, (section 8.2).

Multi-head attention: Each encoder layer uses self-attention and feed-forward networks. Multi-head attention derives Q, K, and V matrices from the same input, unlike cross-attention.

Add & Norm: Residual connections add the input to each layer's output, followed by layer normalization. This stabilizes training and helps to mitigate the exploding/vanishing gradient problem.

Decoder

Output embeddings: Similar to input embeddings but shifted to the right in the input sequence, enabling the autoregressive generation of tokens. Positional embeddings are added, like in the encoder.

Masked multi-head attention: Prevents the model from attending to future tokens by masking, ensuring each token only considers itself and previous tokens.

Multi-head cross-attention: Allows the decoder to attend to encoder outputs (which is projected as K, V) and its previous layer outputs (projected as Q), enabling interaction between input and output sequences. Sometimes it is called "Encoder-Decoder Attention".

Linear & Softmax: The decoder output passes through a linear layer and softmax to produce a probability distribution over the words' vocabulary. The most probable token is selected and fed back for the next step. In more advanced models, the output sometimes is not the most probable to allow more creative outputs. This is controlled by a 'temperature' parameter in the softmax function.

F Vision Transformer (ViT)

The Vision Transformer (ViT) [20] by Google uses transformers for image classification, leveraging the encoder portion of the transformer architecture. While not a video synthesis model, its foundational principles influence many T2V models.

Traditional CNNs extract local spatial features but struggle with long-range dependencies. ViT addresses this by using transformers' attention mechanism, capturing global dependencies. It consists of patch embeddings, positional embeddings and a transformer's encoder.

In their paper, they found that ViT scale more effectively for visual recognition than traditional CNN networks.

Patch embeddings: The input image is divided into non-overlapping patches (e.g., $16 \times 16 \times 3$), flattened into vectors, and linearly projected into fixed-size vectors (D):

$$\text{Projected} = \text{FlattenPatch} \times W + b$$

Position embeddings: Each patch is assigned a spatial position encoded using sinusoidal embeddings (section 8.2).

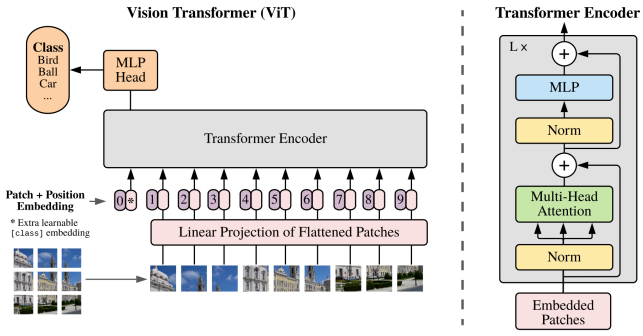


Figure 93: Vision Transformer (ViT) architecture [20]. The first patch + position embedding is the CLS token (classification token) which is used for image classification (its value is changed when the model learns to classify images) [20].

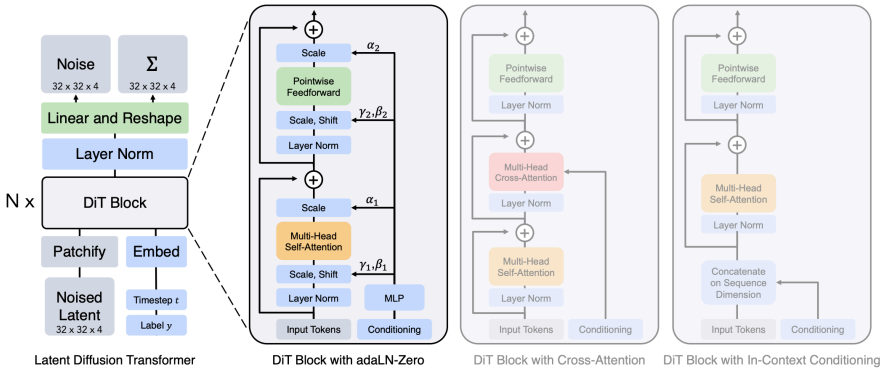


Figure 94: Diffusion Transformer (DiT) architecture [58]. *Right*: three DiT blocks: with adaLN (adaptive layer normalization), cross-attention, and in-context conditioning [58].

Combining the embeddings: The patch embeddings and the positional embeddings are summed together element-wise (**and not concatenated**, channel-wise) to form the input to the encoder.

Transformer encoder: The patch embeddings and positional embeddings are fed to the transformer encoder. It uses self-attention to weight the importance of each patch embedding in order to capture long-range relationships between patches.

Classification head: An MLP layer is used to classify the image based on the encoder's output.

ViT effectively models relationships across image patches, offering a scalable alternative to CNNs. It is success has influenced the use of transformers in image and video synthesis.

G Diffusion Transformer (DiT)

Diffusion Transformer (DiT), introduced in 2023 [58], is a diffusion model based on transformers. It replaces the U-Net in LDMs with a transformer for latent denoising. Leveraging transformers' scalability capabilities, DiT reduces computational complexity (measured in Gflops) and improves sample quality, achieving a state-of-the-art FID score of 2.27 on ImageNet.

Inspired by Vision Transformer (ViT) [20], DiT processes images by dividing them into patches, embedding each as tokens with added positional encodings, and feeding these into the transformer blocks.

In-context conditioning block: Class labels and diffusion time embeddings are added as conditioning tokens to the input sequence, similar to the `[CLS]` token in ViT (added at the beginning of the input sequence). They are removed after processing, and the transformer decoder can perform the noise prediction task.

Cross-attention block: In similar manner to LDMs, they use cross-attention with the class label c and diffusion timestep embeddings t and feed it into cross-attention, where the other input sequence (sequence of patch + position embeddings) is used as the query.

Noise prediction & diagonal covariance matrix: The output of the model is learned noise prediction and a diagonal covariance matrix $\sum_\theta$ (shown in figure 94). We optimize $\sum_\theta$ in order to optimize the decoder $\mathcal{D}_{KL}$.

H Diffusion models samplers

Sampling from diffusion model can take thousands of denoising steps. In this section we will see common samplers used in the literature to sample from diffusion models for faster inference. First we need to establish basic mathematical grounds with the forward and reverse diffusion process as described in the DDPM paper [31] in the next two sections.

Forward process

In the forward process we can take step-by-step approach (adding noise at each step, current step depends on previous step only):

$$q(x_t|x_{t-1}) := \mathcal{N}\left(\sqrt{\frac{\alpha_t}{\alpha_{t-1}}}x_{t-1}, \left(1 - \frac{\alpha_t}{\alpha_{t-1}}I\right)\right)$$

where $\alpha \in (0, 1]$ is the scaling factor of the noise scheduler, α_t is the noise added at timestep t.

However, in the DDPM paper [31] they showed that we can go from x_0 (the original image without any noise added) to x_T (pure Gaussian noise) (or to that matter any arbitrary step t) in one step:

$$q(x_t|x_0) := \mathcal{N}\left(x_t; \sqrt{\bar{\alpha}_t}x_0, (1 - \bar{\alpha}_t)I\right)$$

where $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$.

Because it is not straightforward to implement, they used the reparameterization trick:

$$x_t = \sqrt{\alpha_t}x_0 + \sqrt{1 - \alpha_t}\epsilon \tag{21}$$

where $\epsilon \sim \mathcal{N}(0, I)$.

Backward process

In reverse diffusion process we go from noisy image and denoise it iteratively to get the original denoised image:

$$p_\theta(x_{0:T}) := p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$

where $p(x_T)$ is the sampled noisy image (Gaussian noise) and we iteratively multiply by the denoising procedure where each step depends on the previous step.

H.1 DDPM / Stochastic Sampler

Algorithm 5 DDPM Sampling algorithm from the DDPM paper [31].

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\quad \mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

The DDPM sampling algorithm is shown in algorithm 5: random Gaussian noise $x_T \sim \mathcal{N}(0, I)$ is first sampled, and then for T iterations we slowly denoise the image:

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right) + \sigma_t z$$

where ϵ_θ is the denoising neural network (that predicts the noise, the U-Net) with the inputs (x_t, t). Then the noise is removed from the image: $\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \right)$.

The last term $\sigma_t z$ is the variance to vary the samples. For instance, if $z = 0$ then every time we generate an image from the same noisy image x_T we get the same result (the same cat for instance). When we reach the final step $t = 0$ then we set $z = 0$ to get the final denoised image x_0 without variance.

H.2 DDIM Sampler

One problem of denoising an image is that there are multiple paths of the denoising steps to take; in two denoising steps, we can reach the same result but in different ways. The denoising diffusion implicit models (DDIM) paper [84] suggest non-markovian deterministic approach to fast infer samples from the diffusion model:

$$q_\sigma(x_{1:T}|x_0) := q_\sigma(x_T|x_0) \prod_{t=2}^{T} q_\sigma(x_{t-1}|x_t, x_0)$$

where $q_\sigma(x_T|x_0)$ is the forward process to noise an image in a single step (equation 21). The term $q_\sigma(x_{t-1}|x_t, x_0)$ describes how to get to noisy image at timestep t from the original image x_0 from any marginal distribution in the middle of the diffusion process given (x_t, x_0) [¶¶¶]:

$$q_\sigma(x_{t-1}|x_t, x_0) = \mathcal{N}\left(\underbrace{\sqrt{\alpha_{t-1}}x_0 + \sqrt{1-\alpha_{t-1}-\sigma_t^2}\cdot\frac{x_t - \sqrt{\alpha_t}x_0}{\sqrt{1-\alpha_t}}}_{\text{Eq. 21: } x_t = \sqrt{\alpha_t}x_0 + \sqrt{1-\alpha_t}\cdot\epsilon}, \sigma_t^2 I\right) \tag{22}$$

In other words, we can achieve $x_1 00$ if we have $x_1 01, x_0$; we can achieve $x_9 9$ if we have $x_1 00, x_0$ and so on.

The underbrace term is the reparameterization trick (equation 21). We can see that this equation is similar, the only difference is the noise term ϵ: instead of stochastically sampling noise from normal distribution, it is now dependent on x_0 and x_T and is deterministic (the term $\frac{x_t - \sqrt{\alpha_t}x_0}{\sqrt{1-\alpha_t}}$).

Another difference between equations 22 and 21 is the σ_t^2 term; the role of this term is to add stochasticity. If we set $\sigma_t^2 = 0$ then we have no variance, and we deterministically traverse every intermediate noisy image x_t.

Using DDIM sampler the **forward process** can be derived from Bayes' rule:

$$q_\sigma(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0) = \frac{q_\sigma(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)q_\sigma(\mathbf{x}_t|\mathbf{x}_0)}{q_\sigma(\mathbf{x}_{t-1}|\mathbf{x}_0)}$$

The **backward process** $p_\theta^{(t)}(x_{t-1}|x_t)$ leverages the knowledge of the forward process ($q_\sigma(x_{t-1}|x_t, x_0)$):

$$p_\theta^{(t)}(x_{t-1}|x_t) = q_\sigma(x_{t-1}|x_t, f_\theta^{(t)}(x_t))$$

But since we don't have x_0 at inference time, we estimate it using our neural network $f_\theta^{(t)}(x_t)$. Our network knows only to predict noise, however we can reach x_0 from x_t with a single step:

$$f_\theta^{(t)}(x_t) := \left(x_t - \sqrt{1-\alpha_t}\cdot\epsilon_\theta^{(t)}(x_t)\right)/\sqrt{\alpha_t}$$

So at each timestep t our neural network has estimation of x_0; the larger the t the more error we have (since the task of directly denoising from complete noisy image x_T is more challenging than denoising from less noisy image, say x_{10}).

We calculate the error for x_t, x_{t-1}, and so on. The noise prediction task becomes more and more easy.

To summarize, the sampling process is described as follows:

$$\mathbf{x}_{t-1} = \sqrt{\alpha_{t-1}}\underbrace{\left(\frac{\mathbf{x}_t - \sqrt{1-\alpha_t}\,\epsilon_\theta^{(t)}(\mathbf{x}_t)}{\sqrt{\alpha_t}}\right)}_{\text{"predicted } \mathbf{x}_0\text{"}} + \underbrace{\sqrt{1-\alpha_{t-1}-\sigma_t^2}\cdot\epsilon_\theta^{(t)}(\mathbf{x}_t)}_{\text{"direction pointing to } \mathbf{x}_t\text{"}} + \underbrace{\sigma_t\,\epsilon_t}_{\text{random noise}} \tag{23}$$

- The first term: the neural network first predicts the noise, and then it uses the noise prediction to derive to x_0 from x_t (possibly with a lot of error / loss).
- The second term: the same noise prediction $\epsilon_\theta^{(t)}$ is added to x_t but with less severity.

[¶¶¶]Thanks in part to a youtube guide for the explanation of DDIM sampling

- The third term: similar to σ_t^2 in equation 22; it adds stochasticity to the sampling process.

תוכן עניינים

סיכום

העבודה בוחנת התקדמות אחרונה בטכניקות למידה עמוקה לחילול תמונות ווידאו, תוך התמקדות במעבר מהתחום הבוגר של חילול תמונות לאתגרים החדשים של חילול וידאו. בחנו מודלים בסיסיים, כולל VAEs, GANs, ומודלים מבוססי DMs. חקרנו את המודלים החדישים ביותר כמו VQ-GAN, LDM, ו-Imagen לחילול תמונות, והמודלים Video-LDM, Imagen-Video, Make-a-Video לחילול וידאו. להלן סיכום קצר של המודלים שנדונו:

- **VQ-VAEs**: מודלים הסתברותיים המשתמשים במשתנים חבויים לשחזור תמונות, המאפשרים אינטרפולציה חלקה במרחב החבוי. VQ-VAE משפר את VAEs על ידי הכנסת קוונטיזציה וקטורית, אשר מבצעת דיסקרטיזציה למרחב החבוי ומקלה על למידת התפלגות ודגימה בצורה יעילה יותר.
- **GANs**: מודלים אדברסריים הלומדים את התפלגות הנתונים על ידי אימון של מחולל שמייצר דגימות ריאליסטיות, ו-"מבחין" שמבחין בין דגימות אמיתיות למזויפות. עם זאת, GANs סובלים מאי-יציבות בזמן האימון בעקבות פונקציית האובדן האדברסרית, מה שמוביל לקריסת מודים ובעיות התכנסות. כתוצאה מכך, השימוש בהם ירד עם עליית שיטות מבוססות דיפוזיה.
- **Stable Diffusion**: מודל הסתברותי הלומד את התפלגות הנתונים על ידי הוספת רעש בצורה איטרטיבית לנתונים, ואז לומד לבטל את הרעש בהדרגה. Stable Diffusion מהווה בסיס לרוב מודלי חילול התמונות והוידאו בזכות היציבות שלו והתוצרים האיכותיים שהוא מפיק.
- **Imagen**: Imagen משתמש בטרנספורמרים לצורך חילול תמונות מטקסט (T2I), ומשלב את היתרונות שלהם עם מודלים דיפוזיים מדורגים וטכניקות של סופר-רזולוציה כדי להגיע לתמונה ברזולוציה גבוהה. ראינו שככל שיש יותר פרמטרים בטרנספורמר, כך מדדי האיכות FID ו-CLIP משתפרים.

לאחר מכן אנו מעבירים את ההתמקדות לחילול וידאו, הבנוי על מודלים מבוססי חילול תמונה:

- **VideoGPT**: מודל מבוסס טרנספורמר היוצר סרטונים על ידי חיזוי פריימים עתידיים בהתבסס על פריימים קודמים. הוא משתמש ב-VQ-VAE כדי לפעול במרחב החבוי במקום במרחב הפיקסלים, מה שיעיל יותר מבחינה חישובית.
- **Video-LDM**: אשר משתמש במודל T2I LDM מאומן מראש, ומרחיב אותו לחילול וידיאו על ידי הקפאת שכבות המרחב והוספת שכבות קשב זמני וקונבולוציות תלת-ממדיות על מנת להתאים את המודל לנתוני וידיאו. הוא פועל במרחב החבוי, ומשלב "מבחין" של GAN כדי לשפר את הקוהרנטיות הזמנית.
- **Imagen-Video**: מודל T2V המבוסס על העבודה הקודמת של Imagen, ומרחיב אותו לחילול וידאו. הוא עושה שימוש בדיפוזיה מדורגת עם שבעה מודלים מבוססי דיפוזיה לשיפור הרזולוציה המרחבית והזמנית.
- **Make-a-Video**: מודל המבוסס על העבודה של DALL-E 2, אשר עושה שימוש בקונבולוציות פסאודו-3D ובשכבות פסאודו-זמניות כדי לאזן בין יעילות חישובית לאיכות הוידיאו, ובכך מתמודד עם העלות הגבוהה של קשב זמני מלא וקונבולוציות תלת-ממדיות.

אתגר מרכזי במודלים של T2V הוא העלות הגבוהה של האימון, כאשר חלק מהמשימות דורשות שימוש במאות מעבדים גרפיים. למרות מאמצים רבים להוזיל את עלות האימון, גודל מערך הנתונים והמורכבות הזמנית נותרים אתגר מרכזי. דחיסה יעילה יותר של ייצוגי וידאו, חקר של בלוקים מרחביים-זמניים יעילים, והאצה של זמני האימון והחיזוי הם חיוניים לעתיד של חילול וידאו ארוך.

שיטות אוטורגרסיביות לחילול וידאו ארוך סובלות מהצטברות שגיאות [56], מה שמוביל לאיכות ירודה יותר בפריימים המאוחרים. בנוסף, רוב מודלי חילול וידאו כיום מסוגלים להפיק רק סרטונים באורך של פחות מ-10 שניות. לסיכום, המעבר ממודלים לחילול תמונה למודלים לחילול וידאו ממחיש את ההתפתחות של יכולות הבינה המלאכותית בתחום. בעוד שמודלים לחילול תמונה הגיעו לרמת בשלות עם תוצרים מציאותיים מאוד, חילול וידאו נותר תחום הדורש גישות חדשניות בכדי להתמודדות עם קוהרנטיות זמנית ואתגרים חישוביים.

האוניברסיטה הפתוחה
המחלקה למתמטיקה ומדעי המחשב

סקירה של טכניקות למידה עמוקה לחילול תמונות ווידאו

עבודה מסכמת שהוגשה כחלק מהדרישות לתואר מוסמך במדעי המחשב
האוניברסיטה הפתוחה
המחלקה למתמטיקה ולמדעי המחשב

מאת
שלומי דומננקו

תחת הנחייתה של **ד"ר מיריי אביגל**

מאי 2025