An exploration and application of Denoising Diffusion Probabilistic Models

Alexandre Dréan[1,2] , **Julien Segonne**[1,2] , **Ryan Kaddour**[1,3]
[1] Polytechnique Montréal
[2] ENSTA Paris
[3] Centrale Méditerranée
alexandred56700@gmail.com, julien.segonne@hotmail.com, ryan.kaddour@polymtl.ca

Abstract

In this project, we use diffusion models, such as the *Denoising Diffusion Probabilistic Model* (DDPM)[Ho *et al.*, 2020] and its improved version *Improved DDPM*[Nichol and Dhariwal, 2021], to generate Pokémon images from random noise. This project is part of a reproduction effort and does not constitute an original contribution.

Our project is structured around three main steps: (1) careful preprocessing of the Pokémon dataset [Brilja, 2022], including augmentation and normalization to ensure diversity and stability, (2) implementation of a U-Net [Ronneberger *et al.*, 2015] neural network, and (3) complete implementation of the diffusion process.

After experimenting with our own code, due to time constraints, we explored OpenAI's reference implementation of improved diffusion models. The images we generated, along with their evaluations by multimodal models, confirm a clear resemblance to creatures from the Pokémon universe. In short, this project allowed us to explore both the theoretical and practical foundations of image generation through diffusion models.

https://github.com/Alexndrs/diffusion-image-Generator

1 Introduction

The generation of realistic images has always been a central topic in computer vision. GANs (Generative Adversarial Networks) [Gonog and Zhou, 2019] have largely and rapidly dominated the field by generating visually impressive results. They were introduced by Goodfellow et al. in 2014 and enable the generation of increasingly realistic images based on the generator/discriminator principle. The generator creates images while the discriminator tries to distinguish real images from fake ones, until the generator theoretically produces images indistinguishable from real ones. However, GANs have several limitations, notably the difficulty of training and the instability of their results.

More recently, aiming to improve the method proposed by Goodfellow et al., a new family of generative models called diffusion models has emerged. These models are based on the following principle: learning to reverse images that have been progressively noised. A trained neural network could then reconstruct data (images, sounds) from random noise. We can theoretically relate this work to Langevin dynamics, a stochastic process modeling the movement of a particle under the influence of random noise and more deterministic forces.

An initial approach was published by Sohl-Dickstein et al. in the article "Denoising Score Matching with Langevin Dynamics" in 2015 [Sohl-Dickstein *et al.*, 2015], where the authors attempted to apply Langevin dynamics ideas to images by progressively degrading them with Gaussian noise. Then, a reverse process makes it possible to recover the initial images using estimated gradients at each step. This approach was taken up and improved by Ho et al. in the paper "Denoising Diffusion Probabilistic Models (DDPM)" [Ho *et al.*, 2020], published in 2020. DDPMs implement the previous method with a U-Net [Ronneberger *et al.*, 2015] architecture and a noise schedule (β_t) to train a model capable of reconstructing noised images.

In this project, we focus on the implementation and understanding of the DDPM diffusion model to generate realistic images in a specific context: a low-resolution Pokémon dataset [Brilja, 2022] (64×64 pixels). This analysis and practical work will allow us to understand all aspects and steps of image generation with a DDPM, from preprocessing to final generation.

This project is part of a reproduction effort and does not constitute an original contribution. Our goal is to draw inspiration from the DDPM methods and its improved version, *Improved DDPM* [Nichol and Dhariwal, 2021], to generate low-resolution Pokémon images (64×64 pixels). This work involves a full understanding of the successive steps structuring the method. We identified three distinct and complementary phases in our study, which structure our project during both the understanding and future implementation stages:

- **Data preprocessing**: preparing input images for training.
- **Implementation of a U-Net neural network**: architecture adapted to learn to predict the noise injected into the images.
- **Full implementation of the diffusion process**: chaining of noise addition and denoising steps.

2 Data Preprocessing

The first step of our model concerns data preprocessing, which plays a fundamental role in DDPMs. Preprocessing is key to the quality of the training samples and even to stability during generation. It is therefore essential to understand the target distribution to create a coherent and normalized structure.

Let us explore the preprocessing conditions imposed by the DDPM method [Ho *et al.*, 2020]. First, the method assumes that the input data follows a zero-centered distribution. This ensures stability during training (with smoother gradients), faster convergence (activations being well-distributed around linear zones), and consistency with the diffusion model to be explored in the next section. This normalization, which the authors chose in the interval $[-1, 1]$, is achieved by linearly translating each RGB color channel (red, green, and blue).

We chose to work with a low-resolution (64×64 pixels) RGB Pokémon dataset. It contains approximately 9762 images from the game *Super Mystery Dungeon* from the Pokémon franchise. This dataset was chosen for its reasonable size and resolution compatible with the first version of DDPM. Pedagogically, the images are easy to interpret due to their clear and consistent visual theme. Finally, it is publicly available (via Kaggle) and not subject to complex rights. All our experiments will be conducted with this dataset.

Figure 1: Example images from the 64×64 resolution Pokémon dataset, used for training the model.

To improve the variety of training examples and aim for better generation capabilities, we applied a data augmentation phase. This phase increases the diversity of input images and helps prevent overfitting.

The transformations applied during augmentation were:

- **Random horizontal flip**
- **Slight rotations (±10 degrees)**
- **ColorJitter (saturation, contrast, brightness)**
- **Addition of random noise (optional)**
- **Zoom**

These were applied using the `torchvision.transforms` library, only during the training phase.

3 UNet Neural Network

To perform training and then image generation, a UNet neural network was employed. This section presents a key component used in the model that tracks the noise level of an image:

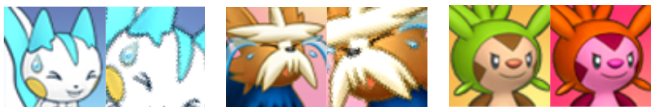


Figure 2: Examples of augmented images (zoom, rotation, flipping, and color modification) from the 64×64 resolution Pokémon dataset, used for training the model.

the time embedding. Then, two versions of a UNet [Ronneberger *et al.*, 2015], a basic one and an improved and optimized one, are described.

3.1 Time embedding

In this model, time embeddings introduce the temporal dependency associated with images: each image fed into the UNet is accompanied by its *timestep*, which corresponds to the noise level in the diffusion process. A sinusoidal positional embedding is first applied to the timestep, then the resulting vector passes through an MLP (two linear layers) to obtain the final time embedding. This embedding is then injected into each residual block so that the model has information about the noise level and can dynamically adapt its transformations.

3.2 UNet

The UNet used in our diffusion algorithm is built in 3 parts:

- The downsampling path (*input_blocks*): consists of an initial convolution to adapt the input dimensions to the model dimensions (*model_channels*), followed by 4 layers each containing 3 residual blocks, one attention block, and a downsampling operation to reduce the resolution (except in the last layer).
- The bottleneck (*middle_block)*: composed of 2 residual blocks separated by an attention block.
- The upsampling path (*output_blocks*): composed of 4 layers each containing 3 residual blocks, one attention block, and an upsampling operation to increase the resolution (except in the last layer).

At the output of the UNet, the result is normalized, the SiLU function is applied to it, and it is projected onto 3 channels. The SiLU activation function is preferred over ReLU partly because it allows more information to pass through in deep layers (negative values are not set to zero, avoiding dead neurons), which is essential since the training images are very rich in information (many details to memorize for the model).

This preprocessing thus allows us to train our diffusion model effectively, which we will detail in the next section.

3.3 Improved UNet

With the initial implemented UNet, the results were not very satisfactory and the model's training time was far too long. We therefore used an Improved UNet presented in the paper [Nichol and Dhariwal, 2021]. The key improvements are as follows:

- Switch to **multi-head attention** with 4 heads and addition of attention blocks for the 8x8 dimensions in addition to those at 16x16. The goal is to capture finer details and smaller patterns.

- New approach to **conditioning** and **injecting** the timestep t into the model: the embedding v of t was initially added to the activations h before normalization.

  $$\text{GroupNorm}(h + v)$$

  In the new model, the embedding is represented by a multiplicative term w and an additive term b which condition the model after normalization. This is known as *Feature-wise Linear Modulation*.

  $$\text{GroupNorm}(h)(w + 1) + b$$

  This improves the stability of the conditioning and allows the model to capture more subtle variations in the noise level over time with this new embedding formulation (multiplicative and additive effect on the normalized activations).
- Use of **gradient checkpointing** to reduce memory usage during training. Instead of storing all activation computations (which are numerous in such a UNet), some activations are recomputed during backpropagation.
- **Dynamic conversion** of the model to float16 (less memory usage due to smaller weight size, faster GPU training) and float32 for important weights (greater precision).

4 The diffusion model

4.1 Modeling

We follow the classical modeling from [Ho *et al.*, 2020], the adopted graphical model is a Markov chain with T timesteps and variables $x_0, x_1 ... x_T$. The initial variable is sampled from the distribution of our training dataset $x_0 \sim q(x_0)$. The noising is done as follows:

$$q(x_t|x_{t-1}) := \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

where β_t is a carefully chosen noise scheduler. By setting $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$, we get a more efficient formula for noising directly from x_0 to any time $t \in [1, T]$:

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

.

Diffusion models like [Ho *et al.*, 2020] aim to reverse this noising phenomenon by successively denoising an image already corrupted to a degree t using another Markov chain p_θ whose graphical model is given by:

$$p_\theta(x_0, x_1 ... x_T) = p(x_T)p(x_{t-1}|x_t)...p(x_0|x_1)$$

and the transition also follows a normal distribution whose parameters θ are the weights of a neural network:

$$p(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

4.2 Choice of schedule

To optimize training, it is crucial to choose the β_t values carefully. A major contribution from [Nichol and Dhariwal, 2021] is the introduction of the cosine schedule, preferred over the linear schedule (where β_t evolves linearly). It is defined by:

$$\beta_t = 1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}$$

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

s represents a small offset to prevent β_t from being too small near $t = 0$. In practice, β_t is clipped at 0.999 to avoid divergence at the end of the diffusion process near $t = T$.

The linear schedule works very well on high-resolution images but much less on 64x64 or 32x32 images. Here, this definition of the cosine schedule is approximately linear in the middle of the process but varies very little at times $t = 0$ and $t = T$ to avoid abrupt noising. It adds noise more gently at the beginning of the process, giving the model more time to learn patterns from less noisy images. It also leaves less training time on highly noised images, which contain fewer distinctive features and patterns from the dataset.

4.3 Towards the definition of a loss function

With the modeling established, we now define an objective function to minimize.

Recall that during training, we have a distribution of initial (non-noisy) data from our dataset: $q(x_0)$. We have a noising technique $q(x_t|x_0)$ and a denoising technique $p_\theta(x_0|x_t)$ given directly by our neural network. Given the true initial image from our training dataset $\hat{x}_0$ that we have noised into $\hat{x}_t$, we aim to maximize its likelihood, i.e.:

$$\min_\theta \mathbb{E}_{t,\hat{x}_0}[-\log p_\theta(x_0 = \hat{x}_0|\hat{x}_t)]$$

Without going into more detail, the paper [Ho *et al.*, 2020] shows that minimizing this objective is equivalent to minimizing:

$$\min \mathbb{E}_{\hat{x}_0,t}[||x_{0,pred} - \hat{x}_0||^2]$$

where $x_{0,pred}$ is the predicted initial image from the neural network given the noisy image at time t. This is also equivalent to training the neural network to predict the added noise. Indeed, for a given x_0, x_t can be written as $x_t = \sqrt{\bar{\alpha}_t}x_0 + (1-\bar{\alpha}_t)\epsilon_t$ with $\epsilon_t \sim \mathcal{N}(0, I)$, and we train the neural network with the objective:

$$\min \mathbb{E}_{\hat{x}_0,t}(||\epsilon_{t,pred} - \epsilon||^2)$$

This latter formulation is the one we use in the code. It's easy to show that it's equivalent to the previous one simply by considering that:

$$x_{0,pred} = \frac{1}{\sqrt{\bar{\alpha}_t}}x_t - \sqrt{\frac{1}{\bar{\alpha}_t} - 1}\epsilon_{t,pred}$$

4.4 Training

Combining everything discussed so far, we can establish a training algorithm for our model.

Algorithm 1 Training algorithm with a batch

Input:
- x_0: training image batch;
- $model$: neural network predicting a noise ϵ with the same dimension as x_0: $X \times [1,T]^{\text{batchSize}} \to X$

$t \sim \mathcal{U}([0,T])^{\text{batchSize}}$
$\epsilon_t \sim \mathcal{N}(0,I)^{\text{batchSize}}$
$x_t = \sqrt{\bar{\alpha}_t}x_0 + (1-\bar{\alpha}_t)\epsilon_t$
$\epsilon_{t,pred,\theta} = model(x_t, t; \theta)$
$L_\theta = ||\epsilon_{t,pred,\theta} - \epsilon_t||^2$
$grad = \nabla_\theta L_\theta$
Perform backpropagation with $grad$ and update the parameters θ of the $model$

4.5 Generation

As described in the modeling section, the denoising process is done progressively, however our model only gives us x_0 from x_t. In this section, we will explain $p_\theta(x_{t-1}|x_t)$ and propose two image generation algorithms: DDPM [Ho *et al.*, 2020] and DDIM [Song *et al.*, 2020].

Given x_t and $x_{0,\theta}$ (predicted by the neural network), we want to define the distribution of x_{t-1}. We can rely on the noising function q to find x_{t-1}:

$$p_\theta(x_{t-1}|x_t) = q(x_{t-1}|x_t, x_{0,\theta})$$

Now, since q follows a Markovian process:

$$\begin{aligned} q(x_{t-1}|x_t, x_{0,\theta}) &= q(x_t|x_{t-1})q(x_{t-1}|x_{0,\theta})\frac{q(x_{0,\theta})}{q(x_t, x_{0,\theta})} \\ &= Cq(x_t|x_{t-1})q(x_{t-1}|x_{0,\theta}) \end{aligned}$$

But:

$$\begin{aligned} q(x_t|x_{t-1}) &= \mathcal{N}(x_t; \sqrt{\alpha_t}x_{t-1}, \beta_t I) \\ q(x_{t-1}|x_{0,\theta}) &= \mathcal{N}(x_{t-1}; \sqrt{\bar{\alpha}_{t-1}}x_{0,\theta}, (1-\bar{\alpha}_{t-1})I) \end{aligned}$$

We show in Appendix A that:

$$q(x_{t-1}|x_t, x_{0,\theta}) = \mathcal{N}(x_{t-1}; c_t^1 x_{0,\theta} + c_t^2 x_t, c_t^3 I)$$

With:

$$\begin{aligned} c_1^t &= \frac{\beta_t\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_t} \\ c_2^t &= \frac{(1-\bar{\alpha}_{t-1})\sqrt{\alpha_t}}{1-\bar{\alpha}_t} \\ c_3^t &= \frac{\beta_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} \end{aligned}$$

For image generation in DDPM [Ho *et al.*, 2020], a denoising step consists of using the model to get $x_{0,\theta}$ and then compute the mean for the next step x_{t-1} as $c_t^1 x_{0,\theta} + c_t^2 x_t$. We also sample a stochastic noise ϵ from $\mathcal{N}(0,1)$, allowing us to sample a value for x_{t-1}:

$$x_{t-1} = c_t^1 x_{0,\theta} + c_t^2 x_t + c_t^3 \epsilon$$

For DDIM [Song *et al.*, 2020], we simply take $\epsilon = 0$ which makes the denoising process deterministic but less so for the noising process.

Algorithm 2 Generation algorithm

Input:
- $shape$: Dimension of the data to generate;
- $model$: neural network predicting a noise ϵ with the same shape: $(shape) \times [1,T]^{\text{batchSize}} \to (shape)$

$t = T$
$x_t = \mathcal{N}(0,I)$ with shape $(shape)$
while $t > 0$ **do**
  $\epsilon_{t,\theta} = model(x_t, t; \theta)$
  $x_{0,\theta} = \frac{1}{\sqrt{\bar{\alpha}_t}}x_t - \sqrt{\frac{1}{\bar{\alpha}_t} - 1}\epsilon_{t,\theta}$
  $c_1^t = \frac{\beta_t\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_t}$
  $c_2^t = \frac{(1-\bar{\alpha}_{t-1})\sqrt{\alpha_t}}{1-\bar{\alpha}_t}$
  $c_3^t = \frac{\beta_t(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}$
  if algo=DDIM or t=1 **then**
    $\epsilon = 0$
  else
    $\epsilon = \mathcal{N}(0,I)$
  end if
  $t = t-1$
  $x_t = c_t^1 x_{0,\theta} + c_t^2 x_t + c_t^3 \epsilon$
end while
return x_t

5 Training and Results

5.1 Some comments on the code...

We manually implemented all the algorithms previously described in the following files:

- diffusion/diffusion.py,
- diffusion/scheduler.py,
- model/unet.py,
- scripts/train.py,
- scripts/generate.py,

We progressively tested our code with unit tests to minimize potential issues. For instance, we ensured—by generating videos—that the schedulers correctly produce Gaussian white noise over time.

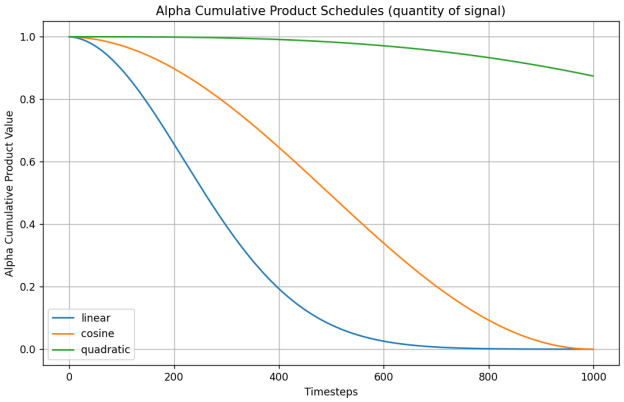
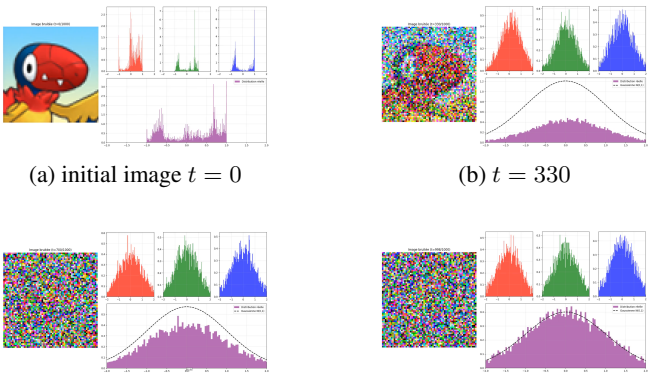


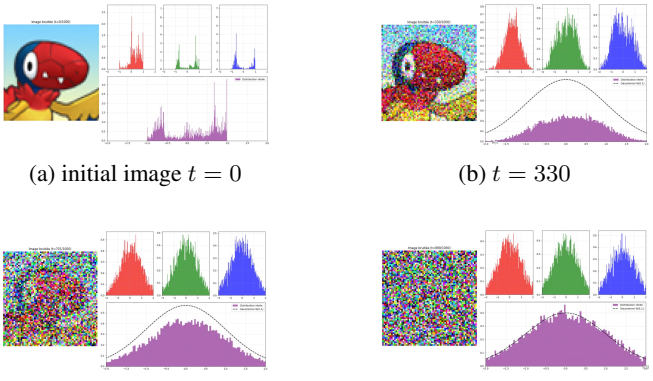
Figure 5: Evolution curves of $\bar{\alpha}_t$ for different schedulers



(a) initial image $t = 0$ (b) $t = 330$

(c) $t = 700$ (d) $t = 998$

Figure 3: Linear scheduler, images and color distributions



(a) initial image $t = 0$ (b) $t = 330$

(c) $t = 700$ (d) $t = 999$

Figure 4: Cosine scheduler, images and color distributions

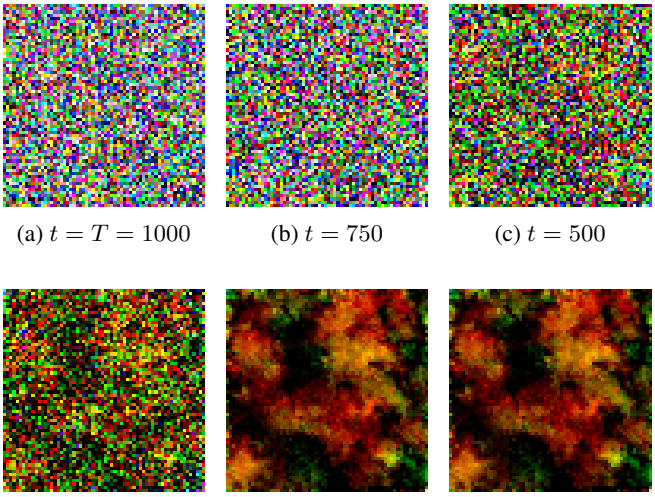
This clearly shows that both of our schedulers—linear and cosine—work as expected, gradually adding noise to the image. The cosine scheduler adds noise more slowly than the linear one, as seen when comparing the curves in Figure 5 or the noisy images generated with the linear scheduler: Figure 3, versus the cosine scheduler: Figure 4.

Once the scheduler was implemented, we coded the UNET [Ronneberger *et al.*, 2015] in model/unet.py.

Finally, we implemented the training and generation (DDPM) functions of the diffusion model by following precisely Algorithms 1 and 2.

After verifying there were no syntax errors in our code, we started training the model for around ten epochs using canonical parameters and the linear scheduler. After about 15 hours of training, we observed that our loss function plateaued and the results were rather disappointing—but not completely discouraging. Indeed, we were still able to generate a denoised image from noise, as shown in the images of Figure 6.

We first suspected the linear scheduler to be the cause of

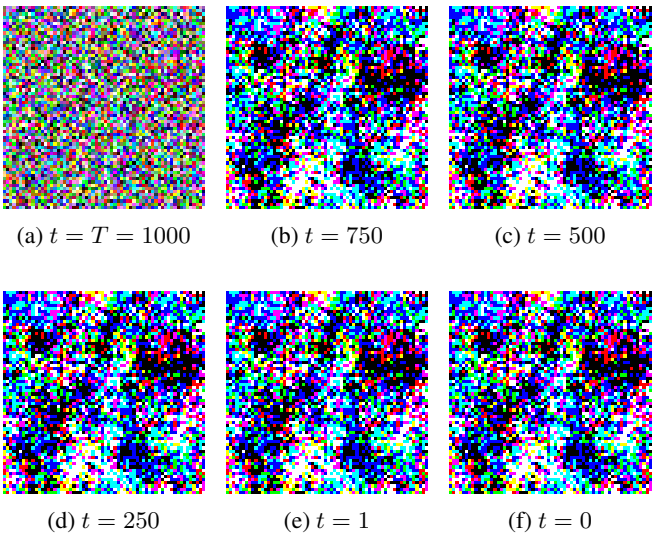


(a) $t = T = 1000$ (b) $t = 750$ (c) $t = 500$

(d) $t = 250$ (e) $t = 1$ (f) $t = 0$

Figure 6: Generation with our model and the linear scheduler

these poor results. Indeed, we believe the linear scheduler was not well suited because our images are small and information is quickly lost during the noising phase, as shown in Figure 3. We therefore tried training the model with the cosine scheduler and obtained even worse results, as shown in Figure 7.



(a) $t = T = 1000$ (b) $t = 750$ (c) $t = 500$

(d) $t = 250$ (e) $t = 1$ (f) $t = 0$

Figure 7: Generation with our model and the cosine scheduler

While writing these lines, I strongly suspect the issue lies in a clipping problem during the generation phase, causing pixel values to diverge—resulting in either very dark or very bright regions. This could explain why the cosine scheduler performs worse than the linear one: at early steps (t large), the cosine scheduler must make larger denoising steps because information is mostly lost near the end, whereas with the linear scheduler there is already little difference at large t. I will try to fix this later, but due to time constraints we chose

to use the official code from the improved DDPM paper [Nichol and Dhariwal, 2021]. The rest of our results were obtained by adapting the code from the following GitHub repository: https://github.com/openai/improved-diffusion. More precisely, we adapted the code in improved_diffusion/gaussian_diffusion.py which re-implements Algorithms 1 and 2 and, unlike our version, uses log-variances to perform the t to $t-1$ transitions and properly clips values at every step to prevent divergence. To significantly speed up training, we also used the code in improved_diffusion/unet.py, which was much more optimized than our own Unet implementation and about 10 times faster.

5.2 Results with improved DDPM

While training the model, we obtained the loss curve shown in Figure 8. We observed that the loss kept decreasing even after 120 epochs, so more compute time would likely lead to better results.

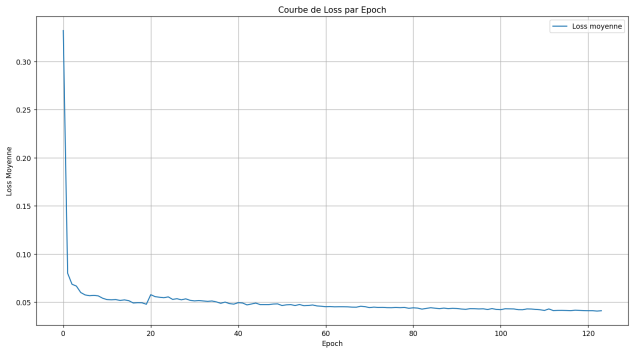


Figure 8: Training Loss evolution (over 120 epochs and 24 hours of training) using cosine scheduler and improved DDPM

Here are some examples of generated images in Figure 9

(a) ... ...

Figure 9: Generation with improved DDPM and the cosine scheduler

5.3 Critique on our results

The generated results resemble the Pokémon from the dataset (few color shades, thick cartoonish lines, sometimes mouths or eyes...) but the analysis cannot go much further. To enrich it slightly, we showed our results to several multimodal language models like ChatGPT, Mixtral, Claude, etc. Here's the prompt and a few interesting excerpts from their answers (full responses are available in Appendix B).

Prompt: "I'm showing you several images. Tell me what you see and what these images remind you of."

ChatGPT (images a,b,j)

"Here's what I see for each image:

First image (purple, smiling character) It depicts a character in purple and pink hues, with a mischievous expression. It reminds me of a **Pokémon** or a cartoon-style creature, ghostly or impish in nature [...]

Second image (green, dynamic) Looks like a green creature or character, with serpentine or dragon-like shapes. It seems dynamic or aggressive, with an open mouth as if attacking or roaring. [...] also feels inspired by **Pokémon** or video games.

Third image (red, intense) This one is harder to clearly interpret, but it shows a bright red entity with high contrast. I see an **eye** and a **mouth**, and it evokes a **powerful creature**, maybe colossal or fiery, like a golem or a mythical bird.

In summary, all three images suggest **stylized creatures**, likely from a game or a **Pokémon-like universe**, Digimon, or a monster card game."

This model clearly recognizes Pokémon-like creatures and uses creature-related vocabulary ("roar", "attack", "mouth", "golem"...), showing that it captures important dataset features.

Claude (all images)

"[...] Overall, these images make me think of abstract art, color field paintings, or perhaps close-up macro photography of textiles or natural materials. They have a dreamy, somewhat psychedelic quality with their vibrant colors and abstract nature. They could also potentially be digital art or AI-generated imagery. [...]"

This model does not recognize characters, focusing mainly on colors and textures, but does correctly guess the images could be AI-generated.

Mistral (images a–h)

"The images you shared appear to be repetitive and abstract patterns, likely generated by artificial intelligence or visual creation algorithms. Here's what I see for each image:

First image: A repetitive pattern with colorful geometric shapes, mainly in pink, violet, and blue tones. It reminds me of crystalline structures or fractals.

[...]

Third image: A pattern with more organic shapes and warmer colors like pink and violet. It reminds me of biological cells or microscopic structures.

Fourth image: A pattern with geometric shapes and green and violet colors. It evokes plant structures or natural formations.

[...]

These images remind me of natural structures or patterns generated by visual creation algorithms. They have an abstract and geometric aesthetic that can be both fascinating and intriguing."

This model doesn't recognize characters or creatures but sees geometric forms and associates them with natural characteristics found in potential Pokémon designs ("plant structures", "crystalline structures"...).

Gemini 2.0 Flash (all images)

"[...] Here are my general impressions and what they remind me of: Overall, the set of images makes me think of a **cartoon** or **video game universe** with a colorful and potentially **fantastical** style. The strong color saturation and often abstract shapes suggest an **imaginary world**. [...]

Image 1: I think I see a rounded shape, maybe a **face** or part of a character, on a red or orange background. It could evoke an **expressive cartoon character**.

Image 2: This image seems to show a **creature** or a green-colored character [...]. Its appearance could be that of a friendly monster or a plant-like character. [...]

Image 6: Purplish tones and more organic shapes. I could imagine a strange flower, an **underwater creature**, or a part of an imaginary landscape.

Image 7: Bright colors [...] with shapes that suggest a character or creature with visible **eyes**. Image 8: [...] with more defined areas that could be **limbs** or body parts. [...]

Image 11: Lighter purple tones and rounded shapes that could evoke a head or a soft, friendly **body**. [...]

In summary, these images give me the feeling of exploring a rich visual universe full of colors and imaginary shapes, likely from a fictional children's work [...], like a cartoon, video game, or illustrated book series. [...]"

This model understands that these are creatures, as it identifies body parts, facial features, and limbs.

A Formula for $q(x_{t-1}|x_t, x_0)$

As we saw in Section 4.5

$$\begin{aligned} q(x_{t-1}|x_t, x_0) &= q(x_t|x_{t-1})q(x_{t-1}|x_0)\frac{q(x_0)}{q(x_t, x_0)} \\ &= Cq(x_t|x_{t-1})q(x_{t-1}|x_0) \end{aligned}$$

Where C is a term independent of x_{t-1}, we can consider it as a normalization constant. The modeling directly gives us:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{\alpha_t}x_{t-1}, \beta_t I) \quad (1)$$

$$q(x_{t-1}|x_0) = \mathcal{N}(x_{t-1}; \sqrt{\bar{\alpha}_{t-1}}x_0, (1-\bar{\alpha}_{t-1})I) \quad (2)$$

Let's transform equation 1 into a distribution over x_{t-1} (Z_1 denotes a normalization term):

$$\begin{aligned} \mathcal{N}(x_t; \sqrt{\alpha_t}x_{t-1}, \beta_t I) &= Z_1 \exp(-\frac{1}{\beta_t}(x_t - \sqrt{\alpha_t}x_{t-1})^2 I) \\ &= Z_1 \exp(-\frac{\alpha_t}{\beta_t}(x_{t-1} - \frac{1}{\sqrt{\alpha_t}}x_t)^2) \\ &= \mathcal{N}(x_{t-1}; \frac{1}{\sqrt{\alpha_t}}x_t, \frac{\beta_t}{\alpha_t}) \end{aligned}$$

$q(x_{t-1}|x_t, x_0)$ is thus the product of Gaussian distributions over x_{t-1}; let's show that this results in a Gaussian distribution and make its mean and variance explicit.

$$\begin{aligned} q(x_{t-1}|x_t, x_0) = CZ_1Z_2 \exp(&-\frac{\alpha_t}{\beta_t}(x_{t-1} - \frac{1}{\sqrt{\alpha_t}}x_t)^2 - \\ &\frac{1}{(1-\bar{\alpha}_{t-1})}(x_{t-1} - \sqrt{\bar{\alpha}_{t-1}}x_0)^2) \end{aligned}$$

Let's work on the term inside the exponential:

$$\begin{aligned} z = &-\frac{\alpha_t}{\beta_t}(x_{t-1} - \frac{1}{\sqrt{\alpha_t}}x_t)^2 \\ &- \frac{1}{(1-\bar{\alpha}_{t-1})}(x_{t-1} - \sqrt{\bar{\alpha}_{t-1}}x_0)^2) \\ &= -\frac{1}{\beta_t(1-\bar{\alpha}_{t-1})} \\ &\times [\alpha_t(1-\bar{\alpha}_{t-1})(x_{t-1}^2 - \frac{2}{\sqrt{\alpha_t}}x_t x_{t-1} + \frac{1}{\alpha_t}x_t^2) \\ &+ \beta_t(x_{t-1}^2 - 2\sqrt{\bar{\alpha}_{t-1}}x_0 x_{t-1} + \bar{\alpha}_{t-1}x_0^2)] \end{aligned}$$

Now,

$$\alpha_t(1-\bar{\alpha}_{t-1}) + \beta_t = \alpha_t - \bar{\alpha}_t + (1-\alpha_t) = 1 - \bar{\alpha}_t$$

Therefore:

$$\begin{aligned} z = &-\frac{1}{\beta_t(1-\bar{\alpha}_{t-1})} \\ &\times [x_{t-1}^2(1-\bar{\alpha}_t) \\ &- 2x_{t-1}(1-\bar{\alpha}_t)(\frac{\alpha_t - \bar{\alpha}_t}{(1-\bar{\alpha}_t)\sqrt{\alpha_t}}x_t + \frac{\beta_t\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_t}x_0) \\ &+ (1-\bar{\alpha}_t)(\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}x_t^2 + \frac{\beta_t\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_t}x_0^2) \\ &= -\frac{(1-\bar{\alpha}_t)}{\beta_t(1-\bar{\alpha}_{t-1})} \\ &\times [x_{t-1}^2 - 2x_{t-1}(c_1^t x_0 + c_2^t x_t) + (c_1^t x_0 + c_2^t x_t)^2] + Q \end{aligned}$$

Where

$$c_1^t = \frac{\beta_t\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_t} \text{ and } c_2^t = \frac{\alpha_t(1-\bar{\alpha}_{t-1})}{(1-\bar{\alpha}_t)\sqrt{\alpha_t}} = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}$$

Q is a constant term (with respect to x_{t-1}), so we can factor it out of the exponential and it becomes part of the normalization constant. In the end, we obtain:

$$\begin{aligned} q(x_{t-1}|x_t, x_0) &= CZ_1Z_2e^Q e^{-\frac{1}{c_3^t}(x_{t-1} - [c_1^t x_0 + c_2^t x_t])^2 I} \\ &= Z' e^{-\frac{1}{c_3^t}(x_{t-1} - [c_1^t x_0 + c_2^t x_t])^2 I} \end{aligned}$$

i.e.:

$$x_{t-1} \sim \mathcal{N}(x_{t-1}; c_1^t x_0 + c_2^t x_t, c_3^t I)$$

with

$$\begin{aligned} c_1^t &= \frac{\beta_t\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_t} \\ c_2^t &= \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} \\ c_3^t &= \frac{\beta_t(1-\bar{\alpha}_{t-1})}{(1-\bar{\alpha}_t)} \end{aligned}$$

Considering that $x_0 = x_{0,\theta}$ depends on the weights of a neural network, we do indeed have a parameterized denoising process $p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t I)$

B Image Analysis by LLMs

We showed our results to several multimodal language models such as ChatGPT, Mixtral, Claude, etc... Here is the prompt and their responses.

Prompt: "I'm showing you several images, tell me what you see and what these images make you think of"

ChatGPT (images a,b,j)

"Here is what I see for each image:

First image (purple, smiling character) It depicts a character in purple and pink tones, with a mischievous expression. It reminds me of a Pokémon or a cartoon-style creature, ghostly or playful, like a trickster spirit.

Second image (green, dynamic) Looks like a green creature or character with serpentine or dragon-like features. The posture seems dynamic or aggressive, with an open mouth as if it's attacking or roaring. It evokes a reptilian creature, possibly inspired by Pokémon or a video game.

Third image (red, intense) This one is harder to discern clearly, but shows a bright red entity with high contrast. We see an eye and a mouth, evoking a powerful creature, maybe colossal or fiery, like a golem or a mythical bird.

In summary, all three images seem to depict stylized creatures, probably from a game or a universe like Pokémon, Digimon, or a monster card game."

Claude (all images)

"I'll share my impressions of each of these images and what they make me think of: Image 1: This appears to be a close-up of a vibrant red fabric or textile with possibly some small objects or decorative elements. The rich red color is quite striking. Image 2: I see what looks like a green textured surface or fabric with some yellow edges/accents. It has an organic, natural appearance. Image 3: This shows what appears to be a colorful abstract image with blues and possibly other colors. It has a somewhat digital or processed appearance. [...] Overall, these images make me think of abstract art, color field paintings, or perhaps close-up macro photography of textiles or natural materials. They have a dreamy, somewhat psychedelic quality with their vibrant colors and abstract nature. They could also potentially be digital art or AI-generated imagery."

Mistral (images a,b,c,d,e,f,g,h)

"The images you shared appear to be repetitive and abstract patterns, probably generated by an AI or visual generation algorithm. Here's a description of what I see for each:

First image: A repetitive pattern with colorful geometric shapes, mainly in shades of pink, purple, and blue. It reminds me of crystalline structures or fractals. Second image: A similar pattern with geometric shapes but with cooler colors like blue and purple. This evokes mineral structures or natural formations. [...] These images remind me of natural structures or patterns generated by visual creation algorithms. They have an abstract and geometric aesthetic that is both fascinating and intriguing."

Gemini 2.0 Flash (all images)

"The images you're showing me are very pixelated and hard to distinguish clearly. However, here are my general impressions and what they make me think of: Overall, the set of images reminds me of a cartoon or video game universe with a colorful and potentially fantastical style. The strong color saturation and often abstract shapes suggest an imaginary world. Here's what I think I see in each image, with the associations they evoke: [...] In summary, these images give me the impression of exploring a rich visual universe full of colors and imaginary forms, likely from a fictional work aimed at children or teenagers, like a cartoon, video game, or illustrated book series. If you can give me more context about these images (where they come from, what they represent), I might be able to give you a more precise interpretation!"

References

[Brilja, 2022] Brilja. Pokémon mugshots from super mystery dungeon, 2022. Accessed on April 4, 2025.

[Gonog and Zhou, 2019] Liang Gonog and Yimin Zhou. A review: generative adversarial networks. In *2019 14th IEEE conference on industrial electronics and applications (ICIEA)*, pages 505–510. IEEE, 2019.

[Ho *et al.*, 2020] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[Nichol and Dhariwal, 2021] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. *arXiv preprint arXiv:2010.02502*, 2021.

[Ronneberger *et al.*, 2015] 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*, pages 234–241. Springer, 2015.

[Sohl-Dickstein *et al.*, 2015] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. pmlr, 2015.

[Song *et al.*, 2020] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.