✨ Member-only story

Comparing and Explaining Diffusion Models in HuggingFace Diffusers

DDPM, Stable Diffusion, DALL·E-2, Imagen, Kandinsky 2, SDEdit, ControlNet, InstructPix2Pix, and more

Mario Namtao Shianti Larcher · Follow
Published in Towards Data Science
33 min read · Aug 24, 2023

Listen Share More

Image generated with Diffusers. Continue reading to discover how and the theory behind.

Table of Contents

Introduction

Embracing the ever-growing interest in Generative AI, including image generation, many excellent resources are starting to become available, some of which I'll highlight below. However, based on my experience, progressing beyond foundational courses demands significant effort, as resources on advanced topics become more scattered.

In this article, we will list the most popular diffusion models from the Hugging Face Diffusers library, which is the primary tool for utilizing this technology. We'll provide brief explanations of these models, compare them, and outline their strengths and weaknesses.

The structure of this article is as follows: we will start by reviewing a few valuable resources for those who are just beginning to study diffusion models. Afterward, we'll provide a brief explanation of the HuggingFace pipelines. Finally, we will delve deep into each pipeline listed in the Popular Tasks & Pipelines section of the Diffusers GitHub repository.

By the end of this article, I hope you will have a solid grasp of the primary diffusion models and associated techniques, and will be in a good position to apply them effectively.

Prerequisites and Suggested Materials

To fully appreciate this article, though I will strive to keep the explanations at an intuitive level, I recommend having a general background in these topics. In this section, I list three resources that I have found useful in my own journey.

Practical Deep Learning for Coders — Part 2

Practical Deep Learning for Coders - Part 2 overview

In this course, containing over 30 hours of video content, we implement the astounding Stable Diffusion algorithm from...

course.fast.ai

Undoubtedly one of my favorite resources, this course not only provides essential insights into diffusion models but also serves as an excellent entry point to acquiring fundamental programming skills in Python and deep learning. Jeremy Howard, the instructor, adopts a highly effective approach by starting with practical applications before delving into the theoretical intricacies. This approach ensures a clear understanding without overwhelming learners with complex mathematical formulas typically encountered in standard courses.

Moreover, the course serves as a seamless continuation from Part 1, requiring no special prerequisites beyond its predecessor. Whether you're a novice or a seasoned learner, this course is a valuable asset in your journey of mastering deep learning and diffusion models.

Hugging Face Diffusion Models Course

GitHub - huggingface/diffusion-models-class: Materials for the Hugging Face Diffusion Models Course

Materials for the Hugging Face Diffusion Models Course - GitHub - huggingface/diffusion-models-class: Materials for the...

github.com

In an article about the Diffusers library, it would be crazy not to mention the official Hugging Face course. This course, which currently has four lectures, dives into diffusion models, teaches you how to guide their generation, tackles Stable Diffusion, and wraps up with some cool advanced stuff, including applying these concepts to a different realm — audio generation.

Generative Deep Learning, 2nd Edition

For book enthusiasts, this is one of my favorite reads on the subject. As the title suggests, this book doesn't just dive into diffusion models; it also covers the broader realm of generative AI. It delves into image generation models like Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs), which are sources of inspiration for diffusion models and used on their side. The second edition covers arguments up until the beginning of 2023, exploring more recent algorithms like DALL·E-2, CLIP, Imagen, Stable Diffusion, and much more.

If you've explored any of these resources or similar ones, you're well-equipped for what lies ahead. If not, you can either go and explore them or proceed in any case with the article; I'll try to keep the explanations as simple as possible, I promise.

Useful Resources

- **MOOC**: Practical Deep Learning for Coders — Part 2
- **MOOC**: Hugging Face Diffusion Models Course
- **Book**: Generative Deep Learning, 2nd Edition

Bonus: I'd like to introduce another resource that I've used to refresh some concepts while writing this article. I'm confident you'll appreciate it as well. If you're interested in understanding AI in a fun, concise, and clear manner, I highly

recommend checking out "**AI Coffee Break with Letitia**". Trust me, it's absolutely worth exploring and subscribing to!

Diffusers Pipelines

What are Diffusers Pipelines?

From the Diffusers documentation:

> *Pipelines provide a simple way to run state-of-the-art diffusion models in inference by bundling all of the necessary components (multiple independently-trained models, schedulers, and processors) into a single end-to-end class. Pipelines are flexible and they can be adapted to use different scheduler or even model components.*

In this article, we will discuss the models behind the most popular pipelines of the Diffusers library. Even though pipelines are meant for inference, the theory behind them is equally relevant for the training of these models. There are a couple of popular training techniques that don't have a dedicated inference pipeline, mainly LoRA and DreamBooth. We will not cover them in this article, but for the latter, I've already written a dedicated article. Feel free to check it out:

Demystifying DreamBooth: A New Tool for Personalizing Text-To-Image Generation

Exploring the technology that turns boring images into creative masterpieces

towardsdatascience.com

How to Use Diffusion Pipelines

Let's learn from a simple example:

```
from diffusers import DiffusionPipeline
import torch

pipe = DiffusionPipeline.from_pretrained(
 "stabilityai/stable-diffusion-xl-base-1.0",
 torch_dtype=torch.float16,
 use_safetensors=True,
 variant="fp16",
)
pipe.to("cuda")

prompt = "A hugging face emoji planet in the solar sistem, detailed, 8k"
```

```
image = pipe(prompt=prompt).images[0]
image
```

This code snippet is all that I used to generate the cover image for this article. We can already observe a few things:

- Even though I am using Stable Diffusion XL, it's not necessary to use the `StableDiffusionXLPipeline` specifically; instead you can use the more general class `DiffusionPipeline`. The `from_pretrained` function will return the correct class object based on the repository ID ("stabilityai/stable-diffusion-xl-base-1.0" in this case) or the path to a local directory.
- It's easily possible, and generally recommended to speed up the process, to change the weight to half precision (float16), specifying the corresponding variant. For example, you can check here that both a `diffusion_pytorch_model.f16` and a non-f16 model exist for the U-Net of Stable Diffusion XL.
- It's advised to use the weights in the safetensors format whenever possible. This format avoids the security issues of pickling and is faster at the same time.
- It's highly recommended to execute this code on a GPU (`to("cuda")`), as diffusion models are computationally intensive. Generating a single prediction often requires around 20–50 forward passes of the model.
- If you were to rerun this code, you would obtain a different result. Diffusion model inference is inherently non-deterministic, implying that each execution produces a diverse outcome (unless you intentionally enforce consistency fixing random seeds, etc.).

In conclusion, as observed, it's remarkably straightforward to utilize these pipelines. This is why I chose to focus on the theory behind them; understanding it at an intuitive level is instrumental in fully harnessing the capabilities of these powerful tools.

Useful Resources

- **Diffusers documentation:** Pipelines
- **Diffusers documentation:** Safetensors

Pipeline: DDPM (Diffusion Models)

Unraveling the Theory

"Denoising Diffusion Probabilistic Models" (DDPM) marked the initial spotlight on diffusion models. While often hailed as the seminal paper on this theme, the concept of diffusion models was introduced back in 2015 within the paper titled "Deep Unsupervised Learning using Nonequilibrium Thermodynamics". The figure below encapsulates the core concept of **diffusion models:**

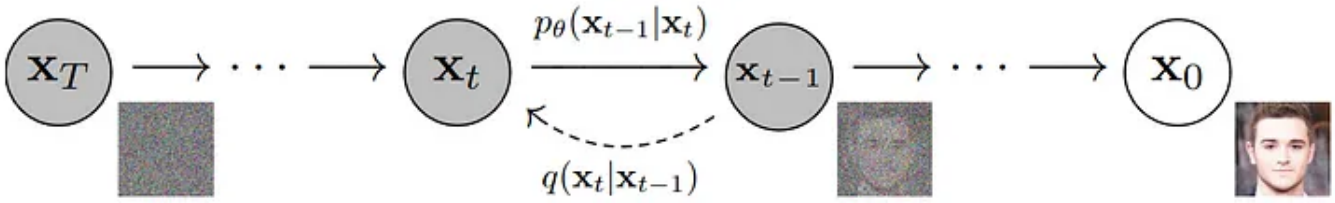


Fig. 2 from Denoising Diffusion Probabilistic Models.

Reading the image from right to left, we observe the **forward** or **diffusion process**, in which we progressively add noise to an image — a straightforward procedure in Python or any other programming language.

Now, imagine having an instrument that can partially remove noise from a noisy image. This tool would facilitate transforming an image composed entirely of noise into a less noisy version, progressing from left to right in the figure above — the **reverse process**. But how do we create this predictive model? According to DDPM, we utilize a U-Net. Given a noisy image, the U-Net works to predict the added noise (or alternatively directly the denoised image). As we introduce the noise ourselves, we have the target variable for free, allowing us to train the model in a self-supervised manner.

The **U-Net** used here isn't the 2015 version; it's a modern adaptation designed specifically for this task. To simplify the U-Net's task, we provide not only the noisy image but also the **timestep** `t` as input. A higher `t` corresponds to a noisier image. This timestep is incorporated into the model using a sinusoidal position embedding, inspired by the Transformer. From the Transformer derives also the self-attention mechanism, tailored for images in this case. Self-attention allows pixels in the 16x16 resolution blocks to attend to all other pixels, boosting the model's capability to generate globally consistent images.

Finally, let's introduce the concept of **sampler** or **scheduler.** According to the Diffusers documentation:

> *The schedule functions, denoted* Schedulers *in the library take in the output of a trained model, a sample which the diffusion process is iterating on, and a timestep to return a denoised sample. That's why schedulers may also be called* Samplers *in other diffusion models implementations.*

In practical terms, schedulers determine the number of steps required to generate the final image and establish the method for converting a noisy image into a less noisy variant, utilizing the model's output. These schedulers can be categorized as either discrete or continuous, as elucidated in the documentation:

> *Different algorithms use timesteps that can be discrete (accepting* `int` *inputs), such as the DDPMScheduler or PNDMScheduler, or continuous (accepting* `float` *inputs), such as the score-based schedulers ScoreSdeVeScheduler or* `ScoreSdeVpScheduler`*.*

In a similar vein, the sampling process can have either a stochastic or deterministic nature.

If you're curious to dive deeper into samplers, that would need a whole separate article. If that sounds intriguing, just give me a shout, and I'll be happy to explore it further!

Applications and Limitations

DDPMPipeline is a pipeline for **unconditional image generation**, hence its practical application is limited compared to other techniques we will explore that allow for greater control over the generated output. Furthermore, the image denoising process using DDPM's scheduler is quite slow; by default, it requires 1000 steps, which translate to 1000 predictions by the U-Net. Given these considerations, the current interest in DDPM is mainly historical, as subsequent works build upon this foundation.

Useful Resources

- **Diffusers documentation**: DDPM
- **Scientific paper**: Deep Unsupervised Learning using Nonequilibrium Thermodynamics (diffusion models)
- **Scientific paper**: Denoising Diffusion Probabilistic Models (DDPM)

Pipeline: Stable Diffusion Text-to-Image

Unraveling the Theory

To date, the primary open-source algorithm for image generation is Stable Diffusion in its various iterations. The initial version of Stable Diffusion is the outcome of a collaboration between CompVis, Stability AI, Runway, and LAION. The primary feature of this model is being a **latent diffusion model** (**LDM**), where the diffusion process occurs not directly in the image/pixel space but within a latent space.

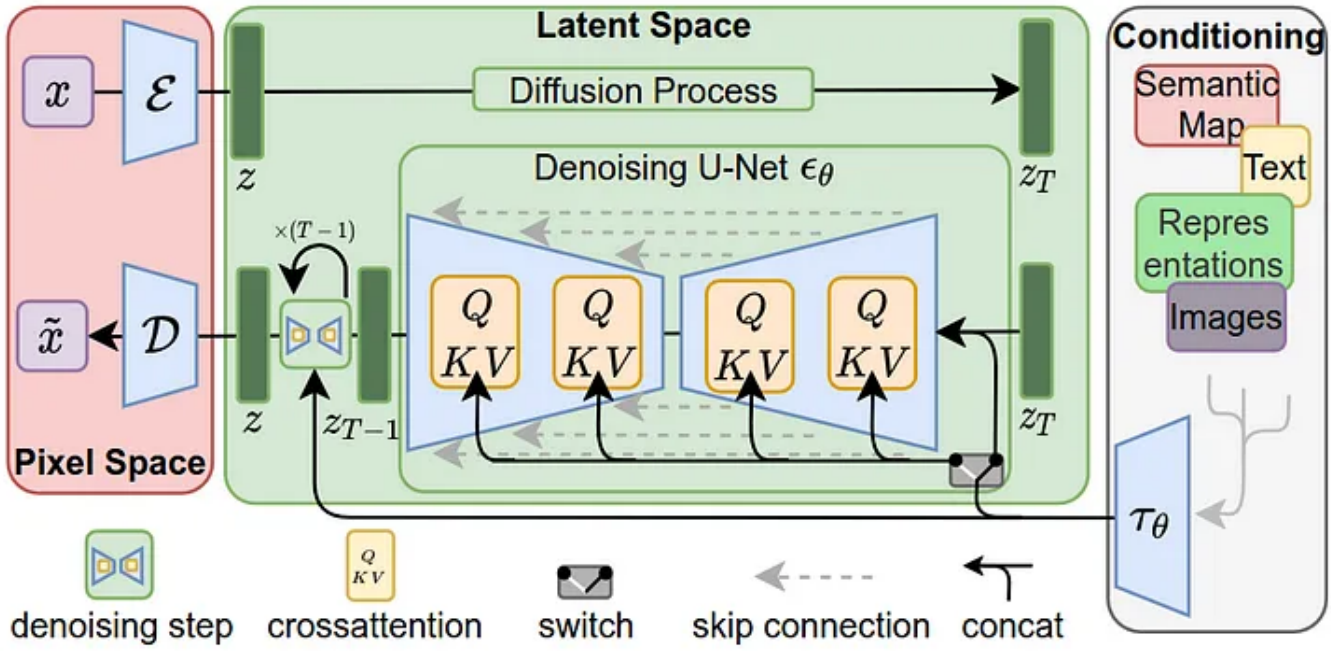


Fig. 3 from High-Resolution Image Synthesis with Latent Diffusion Models.

In practice, before being fed into the U-Net, the image is compressed into a latent space using a **Variational Autoencoder** (**VAE**). After the denoising process, the latent representation is transformed back into an image using the decoder of the same VAE.

Another crucial point to underline is Stable Diffusion's capability to take **textual prompts** as input, partly controlling the generated content. The text is first embedded using a Transformer-based model and then mapped into the U-Net using a cross-attention mechanism. Specifically, Stable Diffusion v1 utilizes the OpenAI

variants.

Stable Diffusion v2 stands out from the original mainly due to a shift in the text encoder to OpenCLIP, the open-source counterpart of CLIP. While one might generally anticipate improved performance in a later version, this assertion is uncertain in the context of Stable Diffusion v2. Notably, **OpenCLIP**'s training on a subset of **LAION-5B**, different from OpenAI's private dataset, along with the use of a highly restrictive NSFW filter, led v2 to noticeably lag behind v1 in representing concepts like celebrities or emulating styles of renowned artists. Some of these limitations were partially addressed in version v2.1, which introduced a less stringent filter and other modifications. For further insight, I found the AssemblyAI article "Stable Diffusion 1 vs 2 — What you need to know" particularly informative.

Finally, Stability AI recently introduced **Stable Diffusion XL** (**SD-XL**), a substantial leap from v2. This iteration competes with leading closed-source models like Midjourney in terms of output quality.

The upgrades of this version include the merging of CLIP and OpenCLIP outputs, the retraining of the VAE with a larger batch size, and the implementation of weight tracking through the **Exponential Moving Average** (**EMA**) technique. The EMA weights can be employed in place of the final weights during inference, leading to a general improvement in performance. This technique aids in reducing some of the overfitting that typically arises in the final iterations, often resulting in the generation of slightly improved weights for inference.

Equally important is SD-XL's effort to address the issues that arise from the use of squared random cropping during training. To enhance this aspect, it employs **cropping parameter conditioning**, which involves informing the model about the parameters that determine how the image has been cropped, similar to what is done for the timestep. This prevents issues like generating headless figures. Simultaneously, the SD-XL version follows modern practices and it's fine-tuned to handle **multiple aspect-ratios** using aspect ratio bucketing. This, along with cropping parameter conditioning, significantly enhances the model's ability to portray horizontal and vertical scenes.

SD-XL also introduces a **refinement stage** where another LDM specialized in high-quality images uses a noise-denoising process introduced by SDEdit, which we will cover in the next pipeline.

Finally, another technique that is not always presented in introductory courses is **offset noise**. The primary reason why we need to modify the initial noise is that, in reality, the image is never completely erased during the forward process (as we perform a finite number of steps). As a result, the model encounters difficulties in learning from pure noise. Quoting from the SD-XL paper:

> *Our model is trained in the discrete-time formulation of [14], and requires offset-noise [11, 25] for aesthetically pleasing results.*

Applications and Limitations

StableDiffusionPipeline (text-to-image) allows the generation of images based on textual prompts. As of today, my recommendation is to use the SD-XL version, which can produce truly astonishing results. Although SD-XL is undeniably exceptional, there are still various cases of failure. The model sometimes faces challenges with very complex prompts that involve detailed spatial arrangements and intricate descriptions. Complicated structures, such as human hands, can still be generated deformed at times. While the photorealism is quite good, it's not yet perfect. Occasionally, a phenomenon known as "**concept bleeding**" occurs, where, for example, a color associated with one element in the prompt is mistaken or extended to another element. The texts generated by SD-XL are significantly better than in the past, but sometimes, especially for longer ones, they contain errors like random characters or inconsistencies. Lastly, it's important to remember that like all generative models, this can inadvertently introduce social and racial biases.

Useful Resources

- **Diffusers documentation**: Text-to-image
- **Scientific paper**: High-Resolution Image Synthesis with Latent Diffusion Models (Stable Diffusion v1, check out my article below, which breaks down this paper for you)
- **Scientific paper**: SDXL: Improving Latent Diffusion Models for High-Resolution Image Synthesis
- **Scientific paper**: Reproducible scaling laws for contrastive language-image learning (OpenCLIP)
- **Scientific blog**: Stable Diffusion 1 vs 2 — What you need to know
- **Scientific blog**: Diffusion With Offset Noise

- **GitHub page:** NovelAI Aspect Ratio Bucketing
- **Appendix:** CLIP

If you're curious to explore the mechanics behind Stable Diffusion further, take a peek at my earlier article:

Paper Explained — High-Resolution Image Synthesis with Latent Diffusion Models

While OpenAI has dominated the field of natural language processing with their generative text models, their image...

towardsdatascience.com

Pipeline: Stable Diffusion Image-to-Image (SDEdit)

Unraveling the Theory

Sometimes we would like to start from a starting image, which can also be made up of our coarse colored strokes, and generate another image that respects the structure of the initial image but whose content is determined by the textual prompt. The simplest technique to achieve this is **SDEdit**, which corresponds to the **Image-to-Image** pipeline.

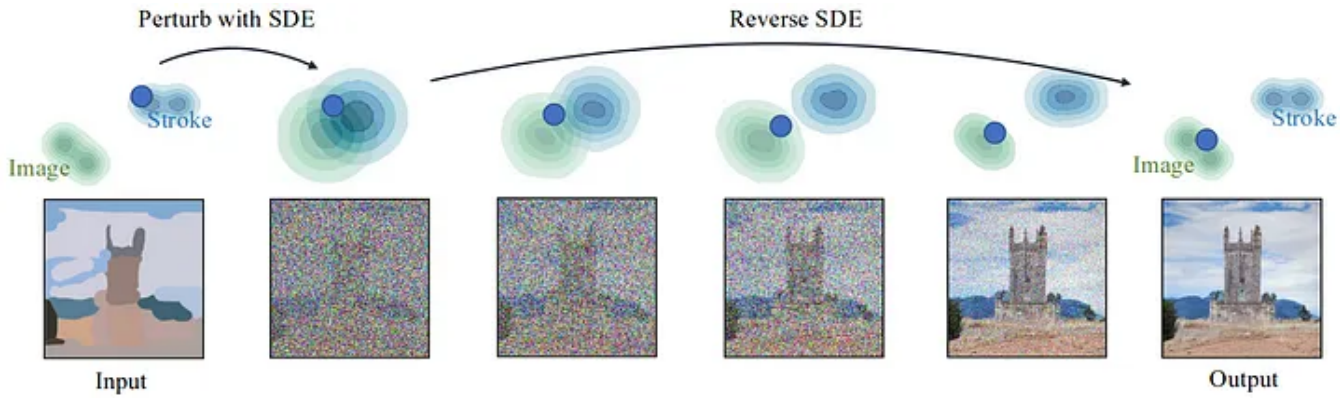


Fig. 2 from SDEdit: Guided Image Synthesis and Editing with Stochastic Differential Equations

During the diffusion process, rather than beginning from random noise, there's nothing stopping us from starting from a later step of the forward process, where we generate the input by incorporating noise based on the chosen starting timestep. As you can see in the figure above, even in the instance of **strokes**, the addition of noise enables the resulting image to be included within the distribution of typical

images. This final point is important as it enable training the model solely with images, but then employing our strokes as input during inference.

It's worth noting that this technique presents a trade-off between **faithfulness** and **realism**, depending on which point of the forward process we start from. In practice, if we use a "strength" parameter equal to 1 during generation, the input image will essentially be ignored, while if "strength" is equal to 0, we will get the same image. The current default is `strength=0.8`.

Finally SDEdit can also be used for **inpainting**, simply by masking the portion of the image that you do not want to modify.

Applications and Limitations

StableDiffusionImg2ImgPipeline is a good pipeline to use when you want to generate an image from some strokes or modify a starting image based on a textual prompt. It's worth noting that the main limitation of this technique is that you can't request significant variations in the structure of the generated image through the textual prompt. The generated image's structure will remain conditioned by the starting structure (unless you choose a strength value very close to 1).

Useful Resources

- **Diffusers documentation:** Image-to-image
- **Scientific paper:** SDEdit: Guided Image Synthesis and Editing with Stochastic Differential Equations

Pipeline: Stable Diffusion Image Variation

Unraveling the Theory

StableDiffusionImageVariationPipeline is a pipeline developed by Lambda which, like Image-to-Image, allows generating variations of an input image.

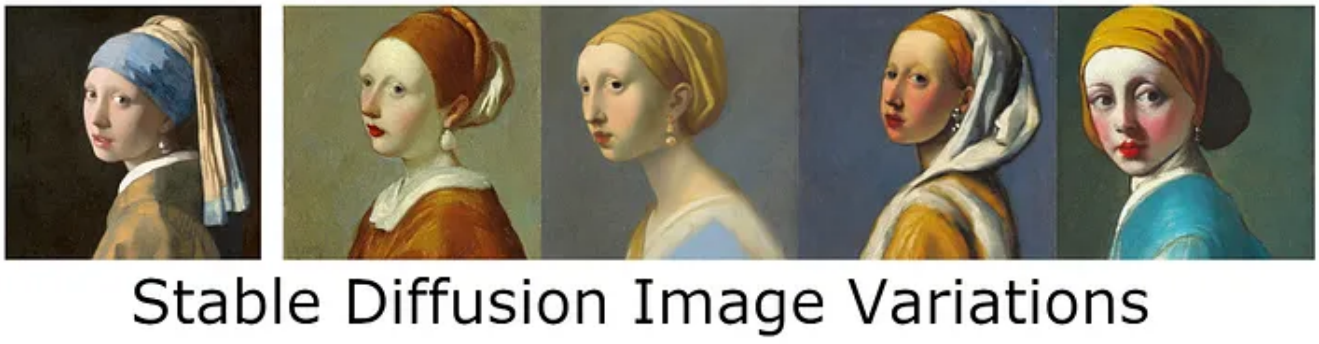


Image from the Stable Diffusion Image Variations Model Card.

Typically, in a task such as text-to-image, generation is conditioned by a textual prompt that is transformed into an embedding through a dedicated encoder. As you can check in Appendix — CLIP, CLIP has two encoders: one for text and one for images. Both of these encoders map the input in such a way that text describing an image has an embedding close to those that don't describe it, and vice versa. This pipeline simply **replaces the CLIP text encoder with the CLIP image encoder**. In this way, generation isn't governed by a textual prompt, but rather by an image that will be decoded by the model not exactly into itself, but into a variant. This is unless the model has overfit to that specific concept and can reproduce it exactly from its latent representation.

Applications and Limitations

This is an interesting pipeline for obtaining images similar to the input. The generated images may not necessarily retain the exact structure of the original image, as with image-to-image approaches, but they might, for example, retain its style or key subject characteristics. One of the main limitations of this technique is that there's not a great deal of control over the generated variations.

Useful Resources

- **Diffusers documentation**: Image variation
- **Model card**: Stable Diffusion Image Variations Model Card
- **Appendix**: CLIP

Pipeline: Stable Diffusion Upscale

Unraveling the Theory

StableDiffusionUpscalePipeline is a **super-resolution** pipeline that enhances the resolution of input images **by a factor of 4**.

Image from the Stable Diffusion x4 Upscaler Model Card.

The method employed, previously introduced in the original Latent Diffusion paper, involves **concatenating the low-resolution image with the latents generated by the VAE encoder**. The model is then trained to generate the high-resolution image based on this input. This model was created by the researchers and engineers from CompVis, Stability AI, and LAION.

Applications and Limitations

The application of this pipeline is quite straightforward: increasing the resolution of an input image.

Useful Resources

- **Diffusers documentation**: Super-resolution
- **Model card**: Stable Diffusion x4 Upscaler Model Card

Pipeline: Stable Diffusion Latent Upscale

Unraveling the Theory

Unfortunately, I didn't find many references about this **latent upscaler** trained by Katherine Crowson in collaboration with Stability AI. In any case, I think it's a safe bet to assume it was trained in a similar way to how the super-resolution model was trained. So, what's the difference? This model, instead of only accepting an image, also accepts latents. It can be used directly on the latents generated in a previous step, without needing to start from an image.

Image by Tanishq Abraham from Stability AI originating from this tweet.

The StableDiffusionLatentUpscalePipeline enhances the resolution of input images **by a factor of 2.**

Applications and Limitations

This pipeline can serve as an alternative to the super-resolution pipeline when we intend to start from latents rather than an image.

Useful Resources

- **Diffusers documentation:** Latent upscaler
- **Model card:** Stable Diffusion x2 Latent Upscaler Model Card

Pipeline: unCLIP (Karlo/DALL·E-2)

Unraveling the Theory

You've probably heard of unCLIP under a different name: DALL·E-2. The version present in Diffusers is derived from kakaobrain's Karlo.

Less describe how the original **unCLIP** work.

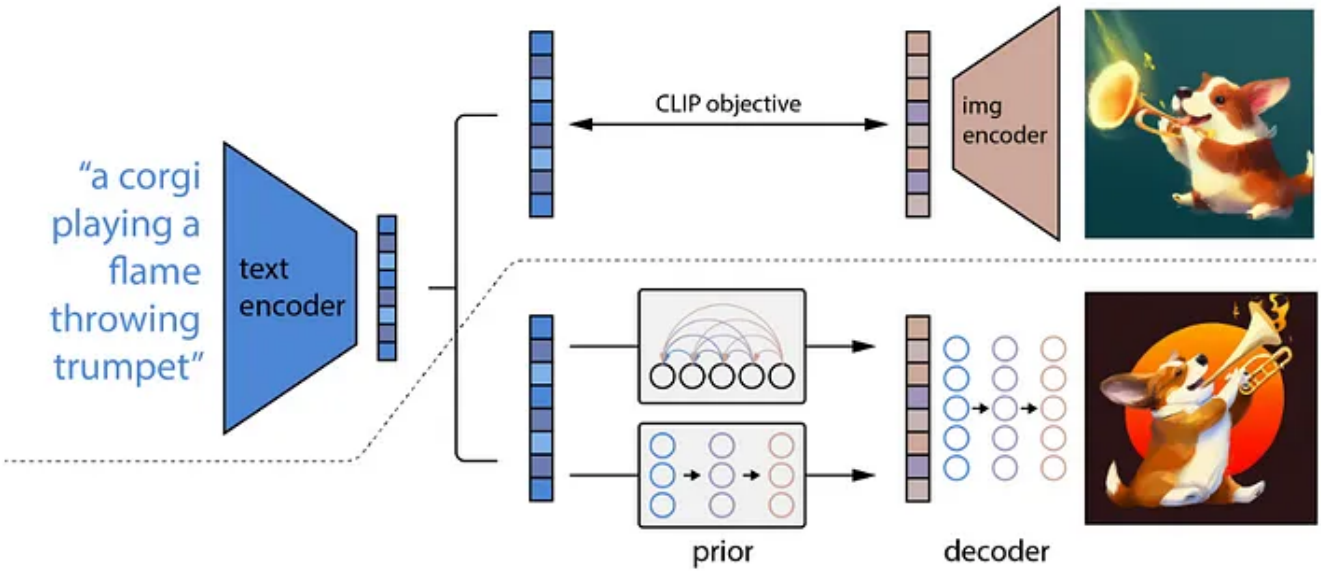


Fig. 2 from Hierarchical Text-Conditional Image Generation with CLIP Latents.

To grasp unCLIP, it's important to know about CLIP. If you're not familiar with CLIP, feel free to check out the Appendix — CLIP. In the image above, the portion above

the dashed line represents CLIP itself. Below, we observe unCLIP. unCLIP employs a model called a "prior" to predict the CLIP image embedding from the CLIP text embedding of the provided prompt. The predicted CLIP image embedding is then input into a decoder, that transforms it into an image. This generated image is subsequently enlarged twice using two upsamplers: first from 64x64 to 256x256, then from 256x256 to 1024x1024.

The paper describes the "**prior**" model as:

> *For the diffusion prior, we train a decoder-only Transformer with a causal attention mask on a sequence consisting of, in order: the encoded text, the CLIP text embedding, an embedding for the diffusion timestep, the noised CLIP image embedding, and a final embedding whose output from the Transformer is used to predict the unnoised CLIP image embedding. […]*

Well, quite confusing, isn't it? First question: **why do we need a prior model at all?** Wasn't CLIP trained to make text embeddings close to their respective image embeddings? Why can't we use them directly? Well, we can, as shown in the paper, but the results will be worse (though not terribly worse). In short, while the embedding of "dog" will be closer to the embedding of the image "dog" than to that of the image "cat", the clusters of text embeddings and image embeddings do not overlap but maintain a gap between them. This phenomenon is quite complex, and if you're interested in delving into it, I suggest you take a look at "Mind the Gap: Understanding the Modality Gap in Multi-modal Contrastive Representation Learning". That said, although we understand that there isn't a strict equivalence between image and text embeddings, where only the same concept in both modes is closer than different concepts, there isn't, in my opinion, a strong theoretical reason why directly using the text embedding shouldn't yield analogous results — it's more of an experimental matter.

Alright, they employed a Transformer instead of a U-Net for this diffusion process (since the aim here is to predict a 1D embedding rather than an image). However, **why they utilized a causal attention mask?** I'm unsure about this, and even the skilled lucidrains who adapted DALL·E-2 to PyTorch doesn't appear to have a solid rationale for it. You can find his response on this topic here.

Another doubt you might have is: **how** on earth do **we input the noised CLIP image embedding** if the CLIP image embedding is exactly what we want to predict? To

answer this question, it's sufficient to remember that we are dealing with an iterative diffusion process, where at the beginning, the noised image embedding will simply be… noise.

Finally, two other tricks.

The first one is to **predict** not one but **two CLIP image embeddings and** then **choose the one closer to the CLIP text embedding**.

The second trick is the use of classifier-free guidance. **Classifier-free guidance** is now a technique used by practically all diffusion models, including Stable Diffusion. During training, it involves occasionally removing text conditioning (in this specific case, 10% of the time). During inference, this means generating one sample with text conditioning and another without it. The difference between the two provides us with the direction in which we want to guide the model (the direction given by our textual prompt). This difference can be used to adjust the sample for the next step in the diffusion process.

The **decoder** is inspired by the architecture of **GLIDE** (**G**uided **L**anguage to **I**mage **D**iffusion for Generation and **E**diting), to which a conditioning based on CLIP embeddings is added. GLIDE, in turn, is inspired by **ADM** (**A**blated **D**iffusion **M**odel), to which it adds text conditioning by encoding the prompt with a Transformer. ADM is an enhanced U-Net with additional attention layers and other enhancements compared to the version used in the paper that introduced popular diffusion models.

The **upsamplers** are also diffusion models (ADM), where **noise is added to the conditioning** through the low-resolution image to make them more robust.

Alright, up until now, we have discussed the original unCLIP/DALL·E-2. However, we pointed out that the implementation found in Diffusers is derived from Karlo. So, **what are the differences between Karlo and DALL·E-2?** The main architectural distinction between Karlo and DALL·E-2 is that **Karlo** includes an enhancement in the super-resolution module designed to upscale from 64px to 256px. This enhancement involves a process consisting of only 7 steps. After the initial 6 steps performed using the standard super-resolution module, the additional super-resolution module is further fine-tuned using a VQGAN-style loss, see Appendix — VQGAN.

Finally, it's important to emphasize that while Karlo shares a very similar architecture, **it is not the original OpenAI's DALL·E-2.** Karlo and DALL·E-2 were trained on different datasets, and there might also be variations in other training details. Consequently, the generated outputs from Karlo could potentially exhibit significant differences in quality when compared to those produced by the original model.

Applications and Limitations

The limitations and applications of unCLIP are more or less similar to those of Stable Diffusion. One additional possibility provided by this model is that tasks like generating **variations** of a starting image become trivial: take an image, pass it through the CLIP text encoder, and decode it through the unCLIP decoder. You might now be asking yourself the question: **Is Stable Diffusion better, or is unCLIP, or other models that we are about to see?**

The answer to this question is not straightforward. First, as of today, **there are no robust metrics to automatically measure the performance of these models.** If you're interested, I can write another article about it, but for now, know that close to metrics like **Fréchet inception distance** (**FID**), the best papers always report **human evaluations** for this reason. Second, as we say in Italy, "Non è bello quel che è bello ma è bello ciò che piace" (what's beautiful is not what's beautiful, but what's liked), meaning that beauty is relative, and different people might prefer the "style" of different models depending on their tastes and the use of the images in question.

Here one data point to let you judge what you prefer between the text-to-image models presented in this article:

From left to right: SD-XL 1.0 Base, Karlo v1 alpha (unCLIP) and Kandinsky 2.2.

I generated the images using the prompt “Astronaut in a jungle, cold color palette, muted colors, detailed, 8k”, choosing my favorite image out of four generations, while keeping all parameters as default. I didn’t include DeepFloyd IF because it requires accepting specific terms and conditions to be used.

In this specific case, in my opinion, the result from SD-XL is the best, closely followed by Kandinsky 2, while the unCLIP output is the least preferable, even considering that the remaining three images, not included here, were significantly worse. It’s worth noting that the default image size of unCLIP (Karlo) is 256x256, whereas SD-XL generates 1024x1024 images and Kandinsky 2 generates 512x512 images (if we use the Diffusers implementations of these models).

As a final disclaimer, please be aware that this test is conducted using only one specific prompt and without utilizing other available parameters to control the generation. Each model possesses unique strengths and can produce outputs that are more or less appealing depending on the subject. Considering that we’re discussing altering just a few lines of code, I highly recommend **experimenting with all of them before determining which one aligns best with your requirements.**

Useful Resources

- **Diffusers documentation:** unCLIP
- **Scientific paper:** Hierarchical Text-Conditional Image Generation with CLIP Latents (unCLIP/DALL·E-2)
- **Scientific paper:** Mind the Gap: Understanding the Modality Gap in Multi-modal Contrastive Representation Learning
- **Scientific paper:** GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models
- **Scientific paper:** Diffusion Models Beat GANs on Image Synthesis
- **Scientific paper:** Classifier-Free Diffusion Guidance
- **GitHub page:** Karlo
- **GitHub page:** DALLE2-pytorch
- **Appendix:** CLIP
- **Appendix:** VQGAN

Pipeline: DeepFloyd IF (Imagen)

Unraveling the Theory

DeepFloyd IF is a model inspired by **Imagen**, a Google text-to-image model.

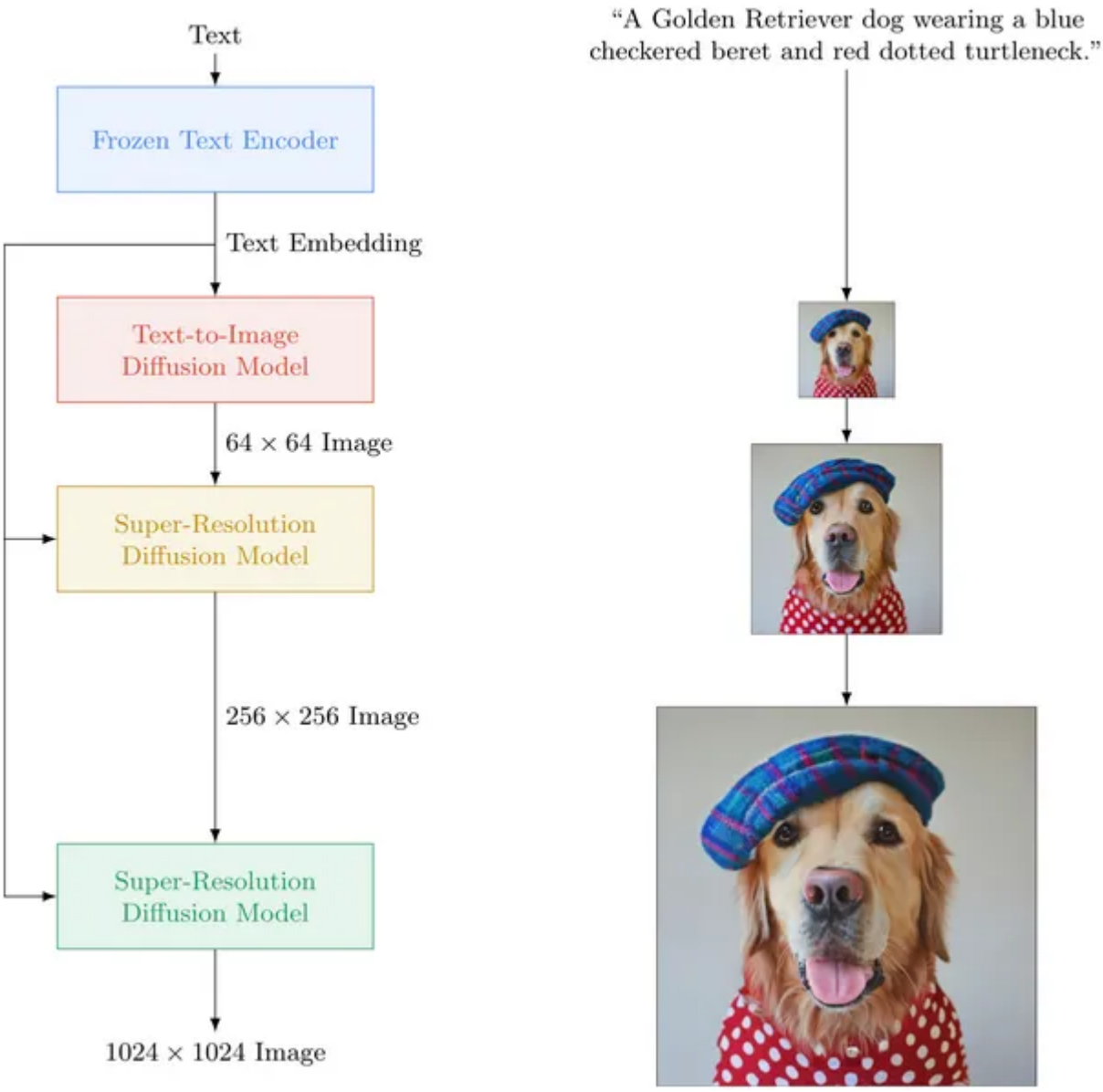


Image from the Google blog post about Imagen.

We have already seen all the elements of these models; both use a text-to-image diffusion model that generates a low-resolution image, 64x64. This image is then upscaled to higher resolutions, first to 256x256 and then to 1024x1024 by other two models.

As the **text encoder**, both models employ a large pre-trained **Text-To-Text Transfer Transformer** (**T5**) by Google, which reframe all NLP tasks into a unified text-to-text format where input and output are always text strings. The text encoder used

appears to be a crucial element in DeepFloyd IF/Imagen, as T5 possesses a broader language understanding compared to CLIP.

Similar to the previously presented models, also in this case, diffusion models are implemented as U-Nets. For super-resolution models, Imagen introduces an **Efficient U-Net**, claimed to be simpler, faster converging, and more memory-efficient compared to prior implementations. The changes made to the U-Net in comparison to previous diffusion models involve "shifting" some of the parameters from high-resolution blocks to low-resolution ones (which have more channels and contain more semantic knowledge respect to initial blocks), using more residual blocks at low resolution, and altering the order of convolution operations concerning up/downsampling. In the case of Imagen, downsampling is performed before convolution, and the opposite is true for upsampling.

Finally, Imagen emphasizes the significance of **classifier-free guidance**. According to the paper:

> *We corroborate the results of recent text-guided diffusion work [16 , 41 , 54] and find that increasing the classifier-free guidance weight improves image-text alignment, but damages image fidelity producing highly saturated and unnatural images [27].*

To benefit from improved image-text alignment without compromising fidelity, two types of thresholding are discussed. The first, previously used by other works, is **static thresholding**, which clips the x-prediction to the range [-1, 1], the same range as the training data x. It is exactly the discrepancy between what the model has seen during training and what it encounters during inference that causes the issue. Static thresholding is essential with large guidance weights but still results in oversaturated and less detailed images as the weight increase. Therefore, the authors introduce **dynamic thresholding**. This technique involves initially choosing a certain percentile of absolute pixel value, let's say 80%. If the value of this percentile, s, exceeds 1 (i.e., more than 20% of pixels are greater than 1 in absolute value), all pixels outside the range [-s, s] are clipped. After this, the values are scaled by s, bringing everything into the range [-1, 1]. Discarding the extreme pixels before normalization helps mitigate the oversaturation problem.

DeepFloyd IF seems to closely resemble Imagen, but without a paper that delves into the details of this architecture, it's uncertain whether there are any significant

modifications I might have missed. According to the authors, DeepFloyd IF outperforms the original Imagen.

Applications and Limitations

DeepFloyd IF can be employed for all the mentioned applications. However, unlike Stable Diffusion and unCLIP, currently, users need to accept the DeepFloyd LICENSE AGREEMENT before utilizing it.

Useful Resources

- **Diffusers documentation**: DeepFloyd IF
- **Scientific blog**: DeepFloyd IF
- **Scientific blog**: Imagen
- **Scientific blog**: Exploring Transfer Learning with T5: the Text-To-Text Transfer Transformer

Pipeline: Kandinsky

Unraveling the Theory

Kandinsky is an AI Forever model that inherits best practices from DALL·E-2 and Latent Diffusion while introducing some new ideas.

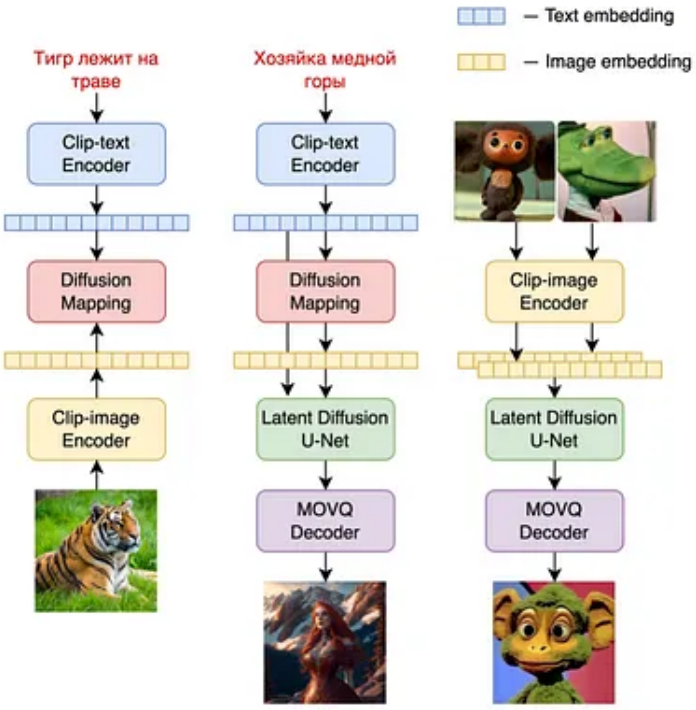


Image from the Kandinsky GitHub page.

Just like DALL·E-2, Kandinsky incorporates a **prior** model (**Diffusion Mapping**) to predict CLIP image embeddings based on CLIP text embeddings. Furthermore, akin to Latent Diffusion, the diffusion model doesn't **operate** in pixel space like DALL·E-2/Imagen, but rather **in the latent space.**

An important difference is that the latest versions, 2.2 and 2.1, of Kandinsky use **XLM-RoBERTa** as the text encoder, thus making the model **multilingual.**

In contrast to DALL·E-2, **the output of the prior model doesn't go directly into a decoder**; instead, it's first directed towards a latent diffusion model.

The **decoder** is **MoVQ**, a model similar to VQGAN (refer to Appendix — VQGAN), which is improved by incorporating **spatially conditional normalization** to tackle the issue of mapping similar neighboring patches to the same codebook index. This addressing prevents the occurrence of repeated artifacts in regions with similar adjacent content. Moreover, the model incorporates multichannel quantization to enhance its flexibility. For the second stage, the autoregressive transformer is replaced with a significantly faster, thanks to its parallel rather than sequential nature, **Masked Generative Image Transformer** (**MaskGIT**).

Applications and Limitations

We have already seen an output from Kandinsky; this model is undoubtedly among the most promising at this moment and competes with the best current available diffusion models. Its usage and limitations are similar to those of Stable Diffusion.

Useful Resources

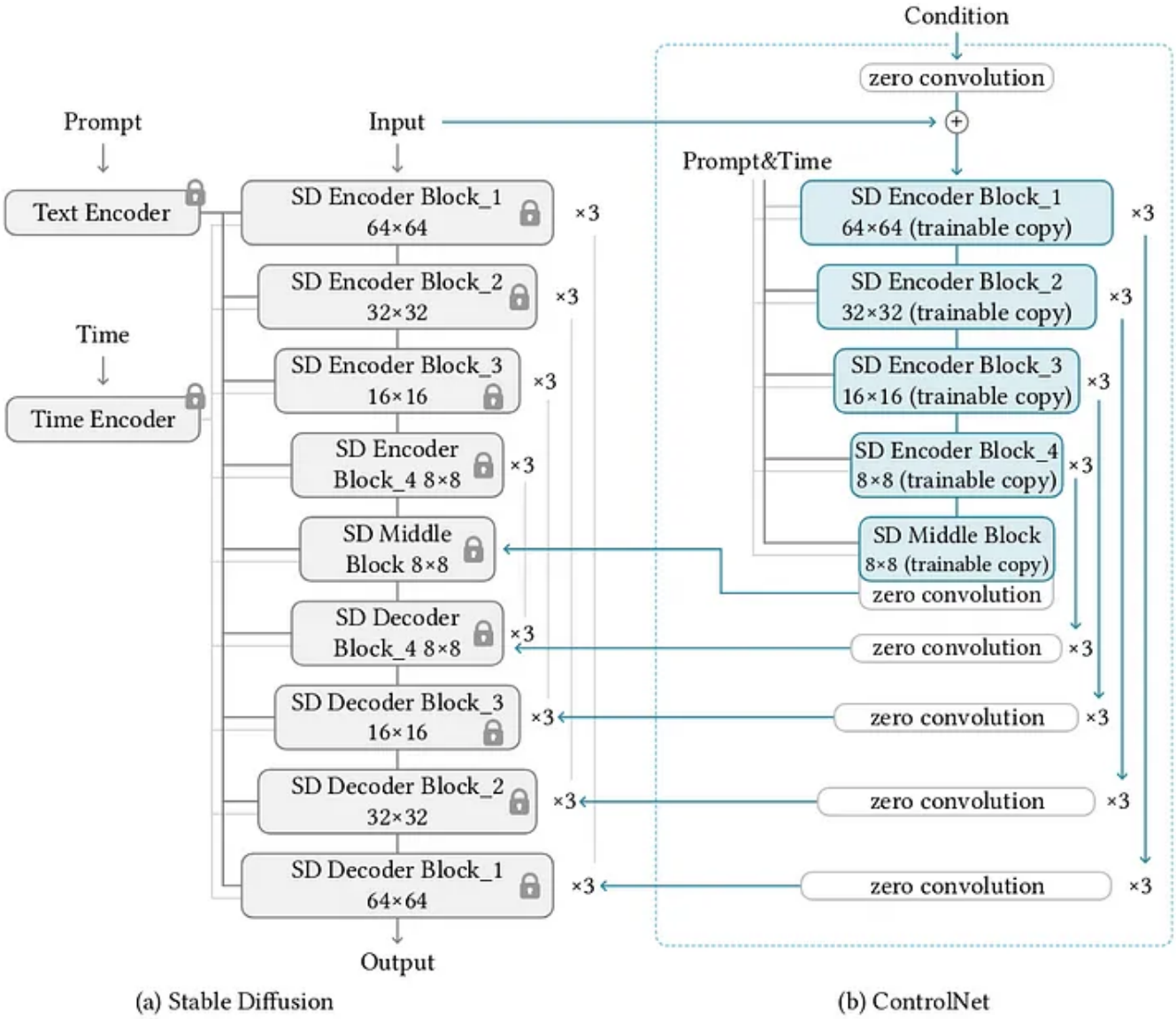
- **Diffusers documentation:** Kandinsky/Kandinsky 2.2
- **Scientific paper:** Unsupervised Cross-lingual Representation Learning at Scale (XLM-RoBERTa)
- **Scientific paper:** MoVQ: Modulating Quantized Vectors for High-Fidelity Image Generation
- **Scientific paper:** MaskGIT: Masked Generative Image Transformer
- **Scientific blog:** Kandinsky 2.1, or When +0.1 means a lot (thanks Google Translate!)
- **GitHub page:** Kandinsky 2.2
- **Appendix:** CLIP

- **Appendix:** VQGAN

Pipeline: ControlNet

Unraveling the Theory

ControlNet is a technique for conditioning the generation of a diffusion model, controlling the structure of what is generated. To some extent, and although the two techniques are complementary, it's like an enhanced SDEdit. The main idea is to automatically generate conditional inputs such as edge maps, segmentation maps, keypoints, etc., and then teach the diffusion model to generate outputs that adhere to the structure of these conditioning inputs. In the original ControlNet paper, Stable Diffusion is used as the base, but this technique can be applied to any model.



Firstly, a copy of the original model is created. The original model is frozen, while the copy is linked to it through a series of zero convolutions.

A **zero convolution** is simply a 1x1 convolution where both the weights and biases are initialized with zero. This type of initialization, along with the fact that the weights of the original model are frozen, ensures that initially, the system is identical to the starting model and only gradually starts to use the conditioning to guide the generation, without forgetting what was originally learned during the extensive training process.

Conditioning involves some form of processing (usually automated) of the input. For instance, we can extract edges from the initial image using a Canny edge detector and teach the model to generate variations faithful to the structure of the original image but with different characteristics that can be guided through textual prompts.

The potential **conditioning inputs** are limited only by our imagination; the authors mention more than a dozen, and over time, the community is inventing new ones. To name a few: edges (e.g., extracted with Canny), human poses (e.g., extracted with OpenPifPaf or OpenPose), semantic maps, depth maps, and so on. Clearly, during training, automation of the extraction is important to speed up the creation of the initial dataset. During inference, there's no restriction on hand-drawing a segmentation maps or even sketching what we want, as it's possible to provide a similar input automatically during training using HED boundary detection and a range of robust data augmentations or alternative techniques, mimicking human sketches.

Applications and Limitations

ControlNet is one of the tools that those who appreciate generative art must have in their toolkit. It's possible to train your own ControlNet from scratch with reasonably limited resources, but often it's not necessary, and you can use ControlNets already trained by the community. The main limitation of this technique is that conditioning often depends on having a starting image with a structure similar to the desired result, or the ability to manually produce an equivalent conditioning. Finally, it's worth noting that it's also possible to combine multiple ControlNets, conditioning, for example, one part of an image on edges and another on a human pose.

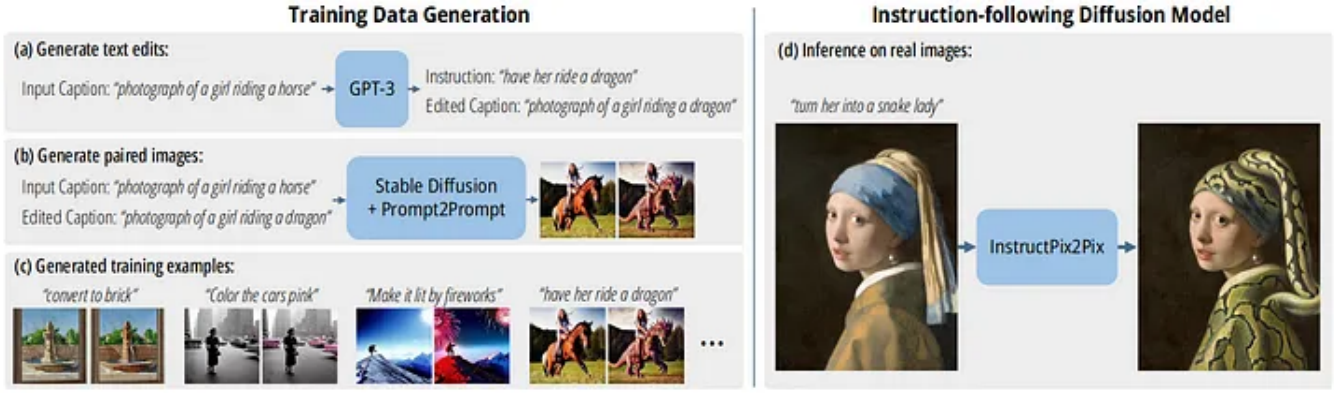
Useful Resources

- **Diffusers documentation**: ControlNet/ControlNet with Stable Diffusion XL
- **Scientific paper**: Adding Conditional Control to Text-to-Image Diffusion Models (ControlNet)
- **Scientific blog**: Ultra fast ControlNet with 🧨 Diffusers

Pipeline: InstructPix2Pix

Unraveling the Theory

InstructPix2Pix is a method for teaching a generative model to follow human-written instructions for image editing.



The method consists of three phases. Firstly, generate a set of input captions, edit instructions, and edited captions. Then, employ another technique called Prompt-to-Prompt (see Appendix — Prompt-to-Prompt) to generate a dataset of image pairs associated with the input and edited captions. Finally, train a generative model to produce the requested modification based on the given instruction.

The **instructions and edited captions**, as shown in the figure, **are generated semi-automatically**. GPT-3, the powerful language model by OpenAI, is fine-tuned on a small sample of LAION captions, to which manually crafted edit instructions and resulting edited captions are added.

At this point, we have all the components to **generate variations of the original images using Prompt-to-Prompt**. An important aspect here is that, depending on the type of instruction given, it may be required for the generated image to remain more or less faithful to the original image. For instance, consider the difference between requesting “make the hair blonde” versus “make it a Miro painting”. Fortunately, Prompt-to-Prompt has a parameter to adjust how much attention should be given to the original image versus the prompt. Unfortunately, this parameter varies case by case.

To address this issue, InstructPix2Pix generates 100 pairs of images for each caption in the training set, varying this parameter. These **pairs are then filtered** using a CLIP-based metric: **directional similarity in CLIP**. This metric measures how consistent the change between two images (in CLIP space) is with the change between the two image captions. Besides enhancing the quality of the generated dataset, this filtering also enhances the model’s robustness to Prompt-to-Prompt and Stable Diffusion failures.

To input the text edit instruction, the authors reuse the same text conditioning mechanism that was initially intended for captions. Meanwhile, for the input image

to be modified, they simply add input channels to the first convolutional layer.

Finally, they employ a form of **classifier-free diffusion guidance** to weigh the image more or less with respect to the text, allowing some control over how closely it adheres to the input image in comparison to following the edit instruction.

$$\begin{aligned}\tilde{e}_\theta(z_t, c_I, c_T) = e_\theta(z_t, \varnothing, \varnothing) \\ + s_I \cdot (e_\theta(z_t, c_I, \varnothing) - e_\theta(z_t, \varnothing, \varnothing)) \\ + s_T \cdot (e_\theta(z_t, c_I, c_T) - e_\theta(z_t, c_I, \varnothing))\end{aligned}$$

Eq. 3 from InstructPix2Pix: Learning to Follow Image Editing Instructions.

Applications and Limitations

InstructPix2Pix is a very useful technique when one wants to modify an image through text without significantly altering elements unrelated to the requested modification. This is different from generating two images where the second one has only a slightly modified prompt. Clearly, this technique doesn't work flawlessly 100% of the time and encounters issues when asked to change the viewpoint, swap object positions, and sometimes, although not as frequently as other techniques, it can lead to unintended excessive changes to the image.

Useful Resources

- **Diffusers documentation**: InstructPix2Pix
- **Scientific paper**: InstructPix2Pix: Learning to Follow Image Editing Instructions
- **Scientific paper**: Language Models are Few-Shot Learners (GPT-3)
- **Scientific paper**: StyleGAN-NADA: CLIP-Guided Domain Adaptation of Image Generators (directional similarity in CLIP)
- **Appendix**: CLIP
- **Appendix**: Prompt-to-Prompt

Appendix

CLIP

The fundamental idea behind **CLIP** is as simple as it is powerful: training two Transformer encoders, one for images and one for text, on a dataset of associated

images and texts to produce similar embeddings when the text refers to the image, and dissimilar embeddings otherwise. With respect to the matrix shown in the figure, the objective is to maximize the sum of scalar products along the diagonal and minimize those off the diagonal:

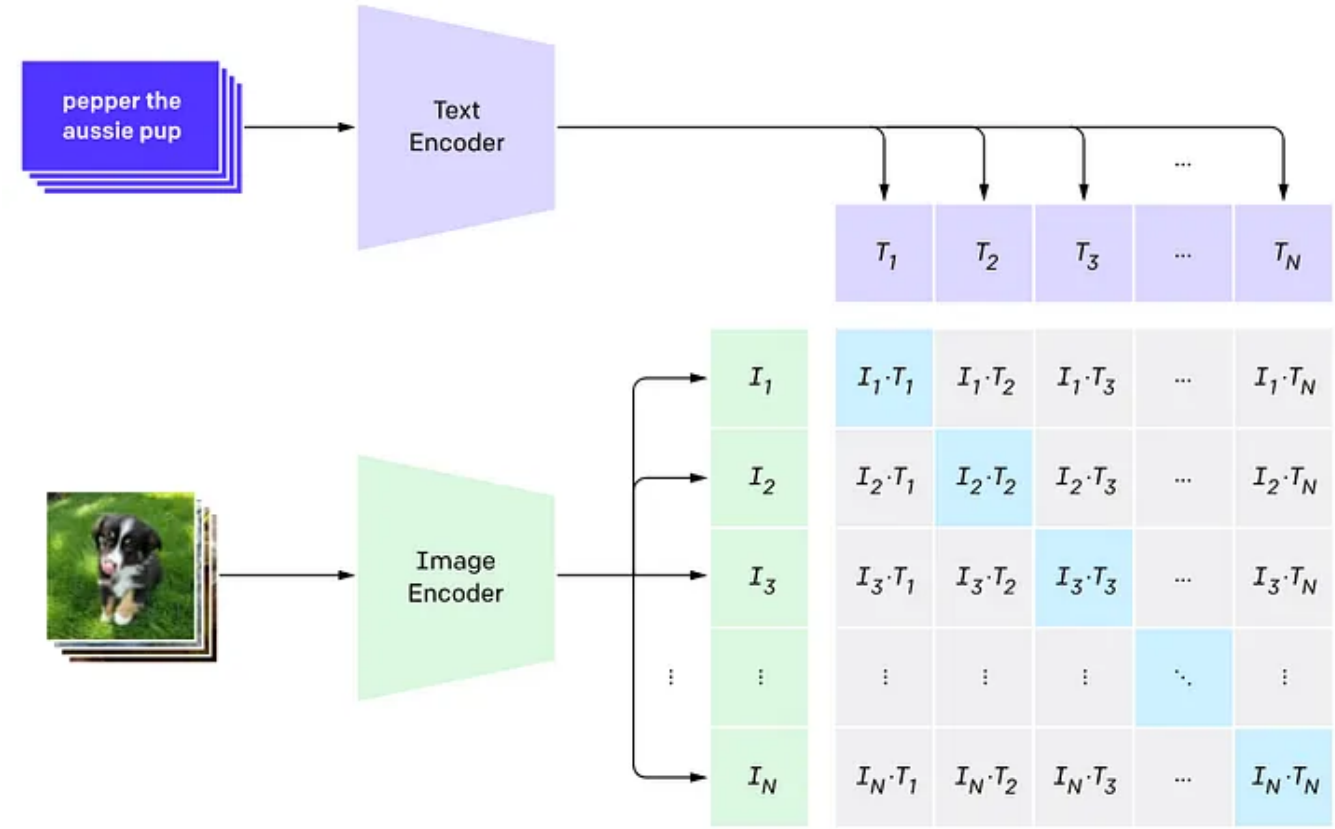


Image from the OpenAI blog post about CLIP.

Since the outputs of the encoders are normalized before scalar products, these correspond to measuring the **cosine similarity** between two vectors, i.e., how much the embeddings "point in the same direction".

Useful Resources

- **Hugging Face documentation:** CLIP
- **Scientific blog:** CLIP: Connecting text and images

VQGAN

In this section, I'll introduce VQGAN and touch on VQVAE.

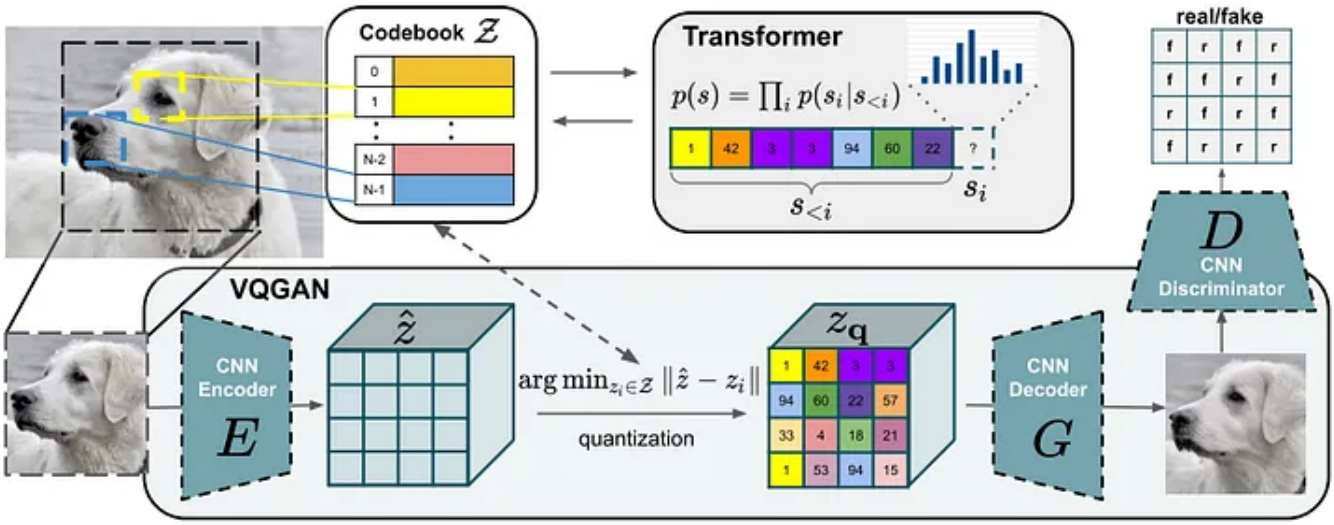


Fig. 2. from Taming Transformers for High-Resolution Image Synthesis.

In the image above, if we consider only *E*, $\hat{z}$, and *G*, what we have is an Autoencoder. VQGAN builds upon **VQVAE** and employs a regularization technique known as **Vector Quantization** (**VQ**). For each spatial position of the encoder output $\hat{z}$, the corresponding vector (whose size depends on the number of channels in $\hat{z}$) is substituted with the nearest vector from a learnable "codebook". This effectively limits the possible inputs to the decoder during inference, allowing them to only be combinations of the learned "codes" and then quantizing the latent space.

The **loss** function employed by **VQVAE**, *L*VQ, comprises three terms.

$$\mathcal{L}_{\mathrm{VQ}}(E, G, \mathcal{Z}) = \|x - \hat{x}\|^2 + \|\mathrm{sg}[E(x)] - z_{\mathbf{q}}\|_2^2 + \|\mathrm{sg}[z_{\mathbf{q}}] - E(x)\|_2^2$$

Eq. 4 from Taming Transformers for High-Resolution Image Synthesis.

The first is a **reconstruction loss**, *L*rec; the second term penalizes the codebook when its elements are distant from the outputs of the encoder. The third term, also called the "**commitment loss**", penalizes the encoder when its output embeddings are distant from the codes in the codebook (we want the encoder to "commit" to a certain codebook).

VQGAN replaces the reconstruction loss with a **perceptual loss.** Specifically, it employs the **Learned Perceptual Image Patch Similarity** (**LPIPS**), which uses a pre-trained VGG16 network to extract features from both the generated and target images, and then calculates the differences between these features.

Second, it introduces an **adversarial** training procedure with a **patch-based** discriminator *D*, aiming to differentiate between real (x) and reconstructed ($\hat{x}$) images:

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

Eq. 5 from Taming Transformers for High-Resolution Image Synthesis.

The complete objective is as follows:

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

Eq. 6 from Taming Transformers for High-Resolution Image Synthesis.

Here, λ represents an **adaptive weight** calculated using the formula:

$$\lambda = \frac{\nabla_{G_L}[\mathcal{L}_{\text{rec}}]}{\nabla_{G_L}[\mathcal{L}_{\text{GAN}}] + \delta}$$

Eq. 7 from Taming Transformers for High-Resolution Image Synthesis.

This weight increases as the gradient of *L*rec (which, for VQGAN, corresponds to the perceptual loss) with respect to the last layer of the decoder intensifies. Conversely, it decreases when the same occurs for *L*GAN. In practice, this means that if *L*GAN is too sensitive to the output of the decoder, its importance is decreased. Conversely, if the perceptual loss (*L*rec) exhibits strong gradients, the importance of *L*GAN is increased, ensuring that a balance is maintained between the two. This approach prevents either term from being entirely ignored when one of them has strong gradients, thus achieving equilibrium between the two objectives.

VQGAN uses a **two-stage approach**. We have already seen the first stage where an encoder, a codebook, and a decoder are learned. In the second stage, as implied by the title of the paper referring to “taming Transformers”, this architecture uses a **Transformer** to predict autoregressively the indices corresponding to codes in the

codebook. Since we know the ground truth indices during training (those generated by the encoder), we can train the Transformer using maximum likelihood. During inference, we don't utilize the encoder (as we don't have an input image, our goal is to generate one), and we leverage the trained Transformer to generate sequences of indices, which are then mapped to codes that the decoder transforms into images.

Useful Resources

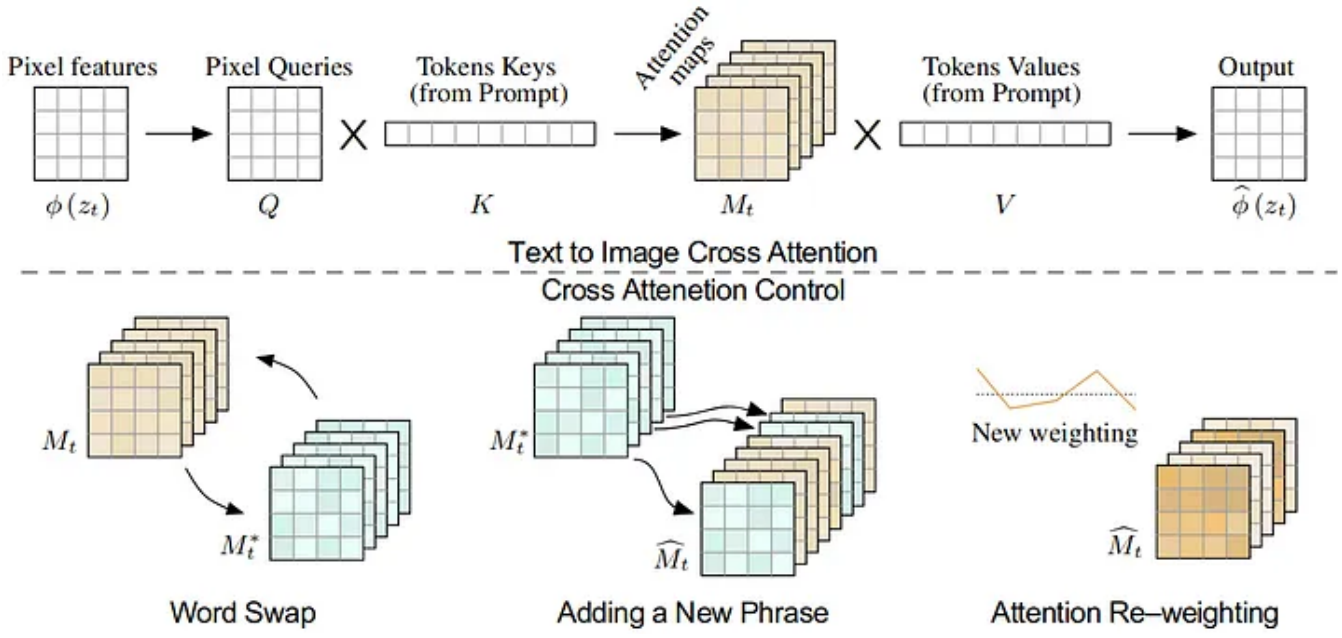
- **Diffusers documentation:** VQModel (VQVAE)
- **Scientific paper:** Neural Discrete Representation Learning (VQVAE)
- **Scientific paper:** The Unreasonable Effectiveness of Deep Features as a Perceptual Metric (LPIPS)
- **Scientific paper:** Taming Transformers for High-Resolution Image Synthesis (VQGAN)

Prompt-to-Prompt

Prompt-to-Prompt arises from a key observation by the authors:

> *we analyze a text-conditioned model in depth and observe that the cross-attention layers are the key to controlling the relation between the spatial layout of the image to each word in the prompt*

Based on this, the method essentially involves **manipulating the cross-attention maps.**



For example, let's say we want to alter an image generated with the prompt "Photo of a cat riding on a bicycle", replacing the bicycle with a car while keeping all other elements as unchanged as possible. In this case, we can generate a new image with the updated prompt but fix the cross-attention maps to those of the previous prompt, where the weights associated with the word "car" become the ones that were originally associated with the word "bicycle".

With this framework, it's now possible to do much more than just simple word replacements. We can use it to give more or less emphasis to a given word, or even add parts to the prompt that weren't present before. In this case, we reuse the attention maps only for the shared parts.

However, keeping the attention maps fixed could overly constrain the geometry of the scene, which, for certain modifications to the prompt, might become too restrictive. To control how much attention is given to the modification and how much of the initial scene geometry is retained, **the injection is limited up to a certain timestep τ.**

$$Edit(M_t, M_t^*, t) := \begin{cases} M_t^* & \text{if } t < \tau \\ M_t & \text{otherwise.} \end{cases}$$

Eq. from p. 7 of Prompt-to-Prompt Image Editing with Cross Attention Control.

This ensures that after capturing the overall composition of the scene in the initial steps, the model can, if needed, alter the geometries in the later steps of the diffusion process.

Useful Resources

- **Scientific paper:** Prompt-to-Prompt Image Editing with Cross Attention Control

Conclusions

Let's summarize what we have covered in this article. To explain what lies behind the most popular pipelines of Diffusers, we have learned about diffusion models, analyzing key ones such as DDPM, Stable Diffusion, unCLIP (Karlo/DALL·E-2), DeepFloyd IF (Imagen), and Kandinsky. Additionally, we've explored techniques for gaining greater control over image generation, like SDEdit, ControlNet, or InstructPix2Pix. To truly understand these techniques, we've also examined important non-diffusion models like CLIP, VQGAN, or techniques like Prompt-to-Prompt (with a pipeline for the latter that might be ready by the time you read this

article). Finally, diffusion model training is somewhat of an art, which is why we have also delved into important tricks like classifier-free guidance, offset noise, CLIP filtering, and so on.

I hope you found this article helpful. Feel free to share your thoughts in whichever way you prefer, I appreciate and consider feedback. If you'd like to show your support, sharing this article on social networks is the best way to do so.

Acknowledgments

First of all, special thanks go to Letitia Parcalabescu of AI Coffee Break. She was invaluable in two ways: first, her videos (check them out, they are great!) were helpful in refreshing or clarifying some concepts for this article; and second, she took the time to read the first draft and provide me with very valuable feedback. Regarding this point, I would also like to express my gratitude to the reviewers at Towards Data Science who are always available for any inquiries and, thanks to their insights, improve the quality of the articles I write. Finally, thanks to you, reading up to this point is no small feat 😊!

Thank you for taking the time to read this article, and please feel free to leave a comment or connect with me to share your thoughts or ask any questions. To stay updated on my latest articles, you can follow me on Medium, LinkedIn or Twitter.

Diffusion Models | Diffusers | Stable Diffusion | Dalle 2 | Deep Dives

Follow

Written by Mario Namtao Shianti Larcher

279 Followers · Writer for Towards Data Science

I like spaghetti code with 🍎 . Currently Head of Computer Vision at Enel Group. AI artist namtao_art on IG/Twitter.

More from Mario Namtao Shianti Larcher and Towards Data Science

Mario Namtao Shianti Larcher in Towards Data Science

Paper Explained—High-Resolution Image Synthesis with Latent Diffusion Models

While OpenAI has dominated the field of natural language processing with their generative text models, their image generation counterpart...

✨ · 10 min read · Mar 31, 2023

389 4

Thu Vu in Towards Data Science

How to Learn AI on Your Own (a self-study guide)

If your hands touch a keyboard for work, Artificial Intelligence is going to change your job in the next few years.

12 min read · Jan 5

2.2K 24

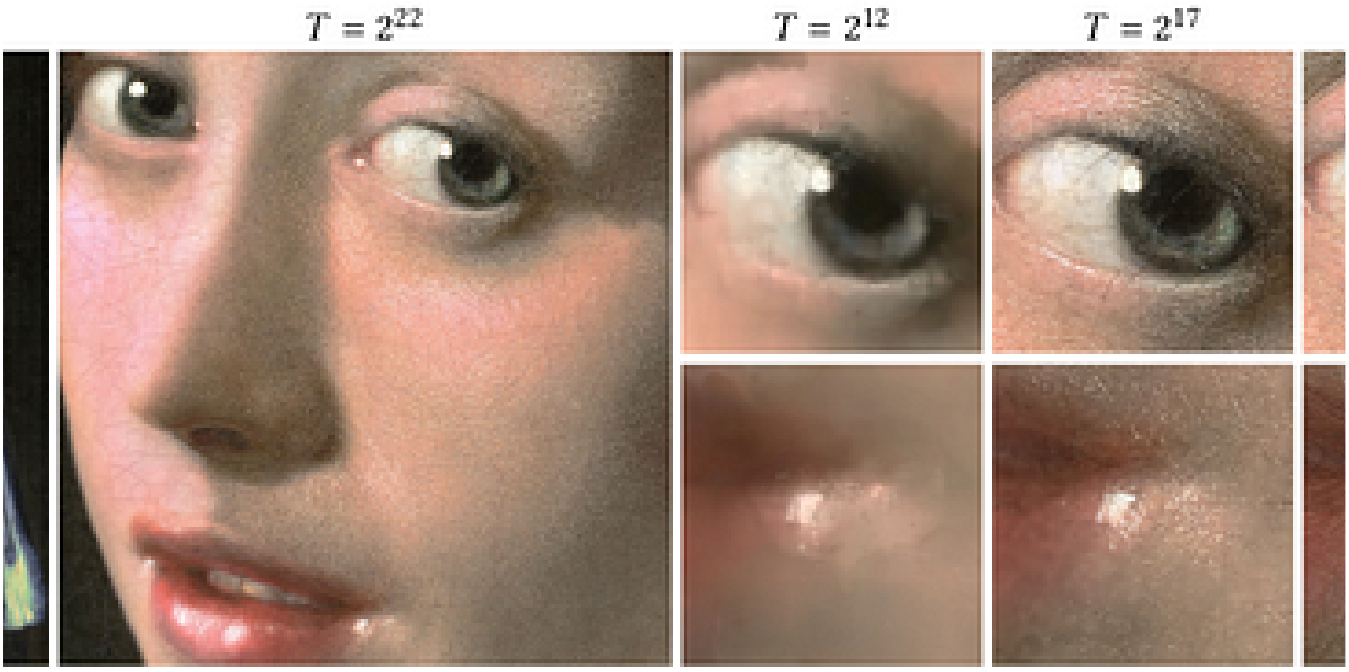
Michael Berk in Towards Data Science

1.5 Years of Spark Knowledge in 8 Tips

My learnings from Databricks customer engagements

8 min read · Dec 24, 2023

1.4K 8



Mario Namtao Shianti Larcher in Towards Data Science

Paper Explained—Instant Neural Graphics Primitives with a Multiresolution Hash Encoding

The seminal NeRF paper has taken the computer vision community by storm, but the proposed approach requires hours of training for every...

· 12 min read · Jul 20, 2022

233

See all from Mario Namtao Shianti Larcher

See all from Towards Data Science

Recommended from Medium

Jonathan Kernes in Towards Data Science

Diffusion Models

What are they, how do they work, and why now?

23 min read · Dec 13, 2022

290 4

Lujia

Three Stable Diffusion Training Losses: x0, epsilon, and v-prediction

In a world drowning in data, the pursuit of creating machines that can understand and process information is relentless. But as any...

3 min read · Oct 9, 2023

53

Lists

Tech & Tools

16 stories · 133 saves

Stories to Help You Grow as a Software Developer

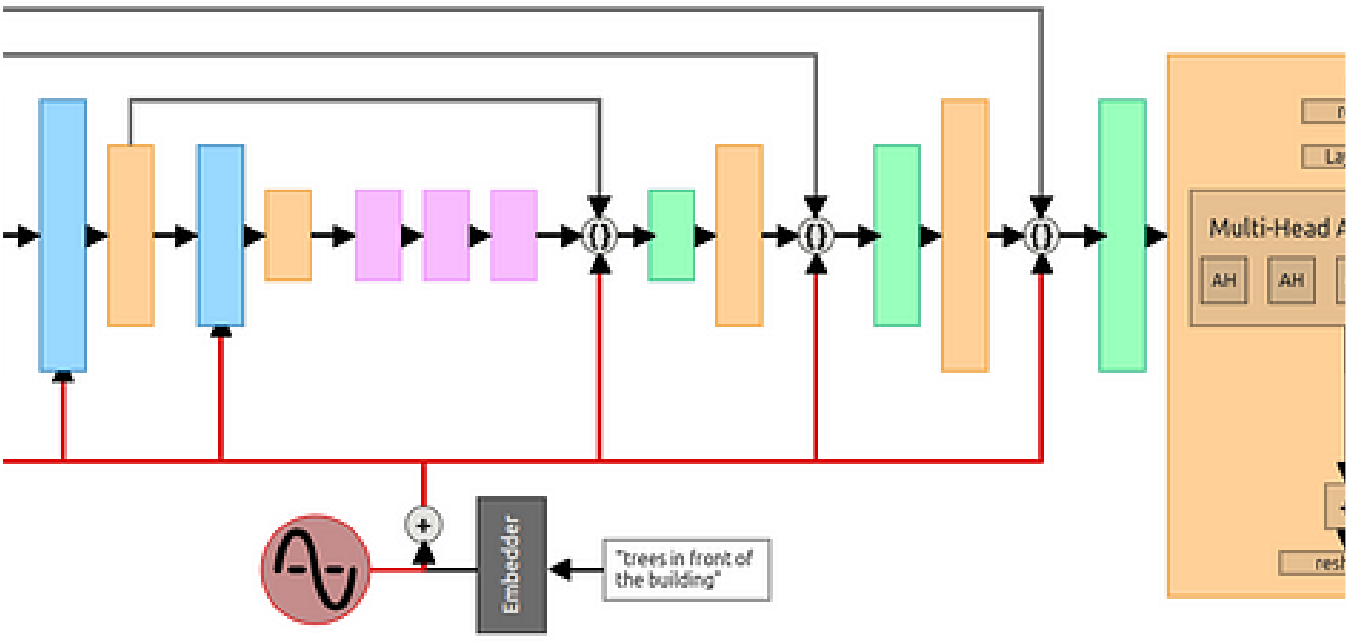
19 stories · 733 saves

Stories to Help You Level-Up at Work

19 stories · 428 saves

Natural Language Processing

1117 stories · 589 saves



Kemal Erdem (burnpiro)

Step by Step visual introduction to Diffusion Models.

How the diffusion models works under the hood? Visual guide to diffusion process and model architecture.

15 min read · Nov 10, 2023

103

Isamu Isozaki

Understanding 3D Diffusion Models

This repository on Stable Dreamfusion mainly inspired this blog.

8 min read · Nov 13, 2023

8

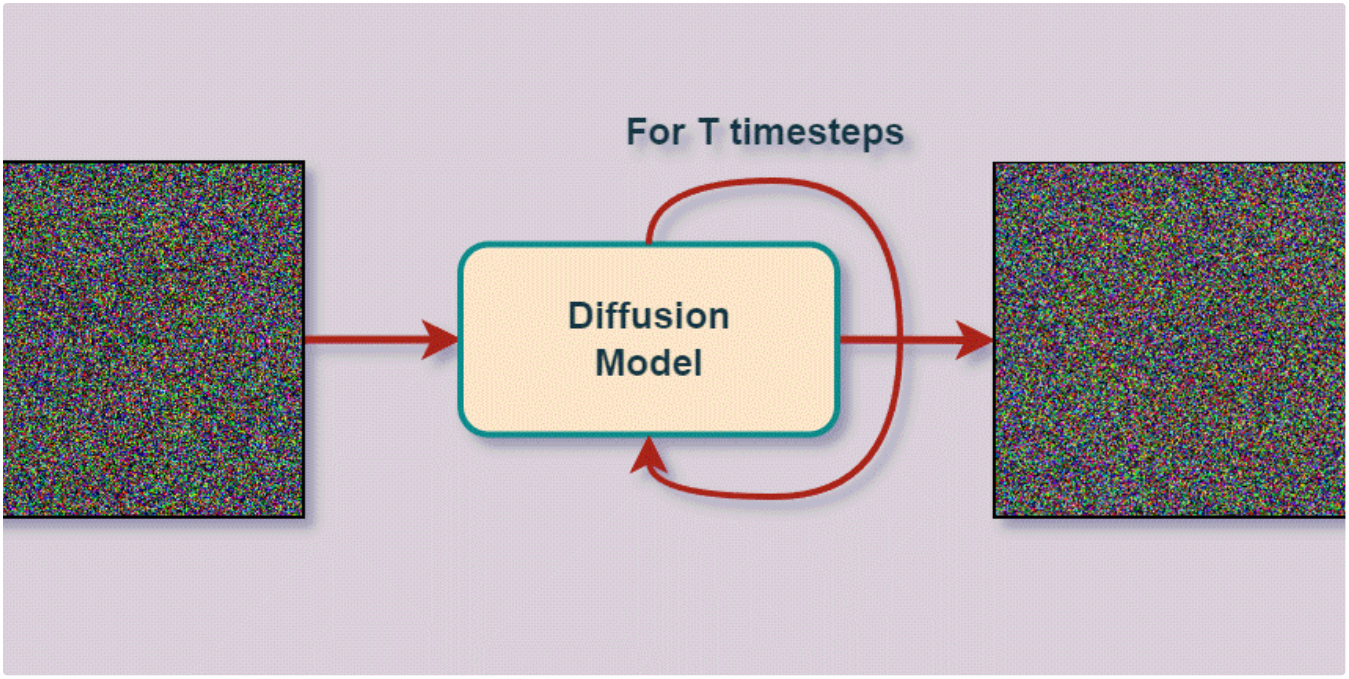
Naman Rastogi

Navigating DDPMs—A Closer Look At Denoising Diffusion Probabilistic Models

In the realm of nature's uncertainty, probability becomes the beacon that illuminates our path towards comprehending reality.

28 min read · Aug 17, 2023

114 1




Gabriel Mongaras in Better Programming

Diffusion Models—DDPMs, DDIMs, and Classifier Free Guidance

A guide to the evolution of diffusion models from DDPMs to Classifier Free guidance

28 min read · Mar 13, 2023


616


8