Stable Diffusion

A latent diffusion model (LDM) is a type of generative diffusion model that learns to generate compressed representations (latents) of images — instead of working directly with high-resolution pixel data.

Traditional Diffusion Models (like DALL·E 1):
- Operate in pixel space (e.g., 512×512×3 images).
- Add noise to an image during training, then learn to remove that noise during generation.
- This is very compute-heavy, especially on high-res images.

Latent Diffusion Models (like Stable Diffusion):
- First compress images into lower-dimensional "latent space" using a VAE encoder.
- Train and sample diffusion steps inside this latent space (e.g., 64×64×4).
- Then decode the result back into an image using a VAE decoder.

Why Use a VAE Encoder in Stable Diffusion?
- You can't directly apply diffusion to full-resolution images (too big).
- So the VAE encoder compresses images into latents.
- These latents still hold semantic meaning (structure, color, objects), but in a smaller size.

Concluding Notes:
- In SD 1.5, the VAE compresses 512×512×3 images into 64×64×4 latent tensors.
- The diffusion process doesn't need to happen on pixels — it can happen on compressed, meaningful features (latents), and still produce stunning results.
- That's the core innovation behind latent diffusion models: do the heavy lifting in compressed space, then reconstruct the final image.

Stable Diffusion In-Depth:

Stable Diffusion is made of **3 main components**:

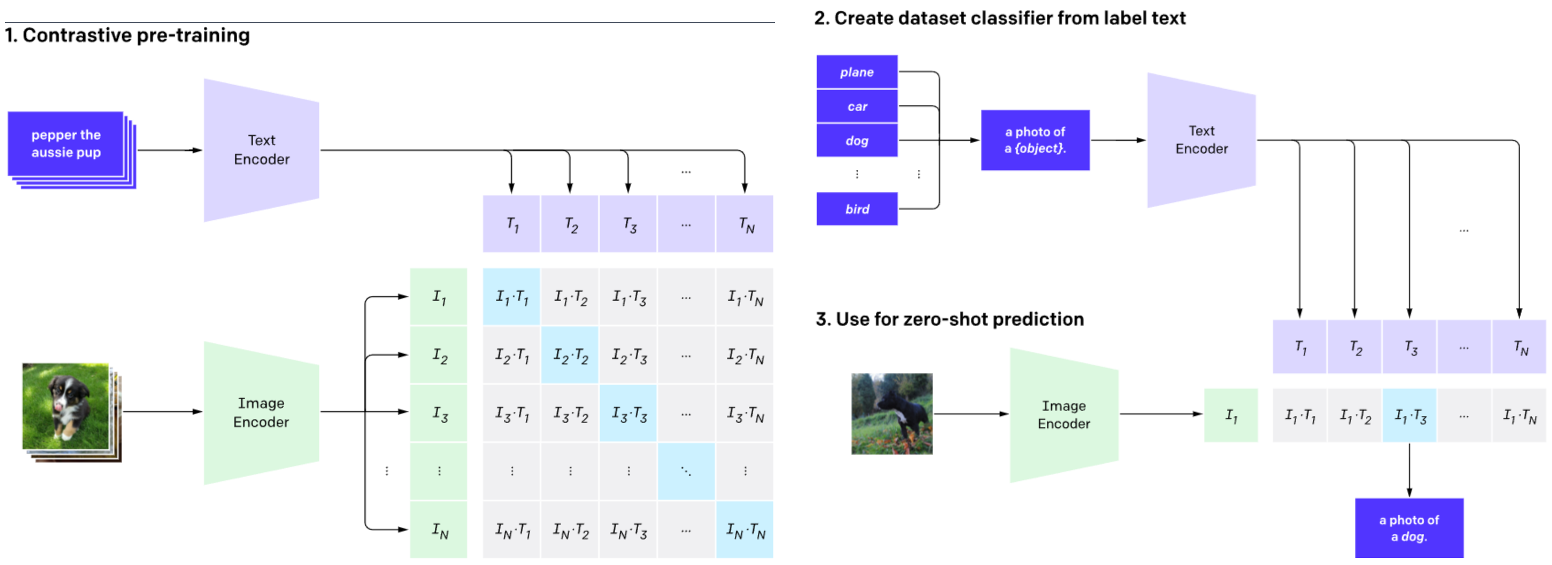
Component	Description
1. Text Encoder (CLIP)	Converts your text prompt into a vector embedding
2. UNet (Denoiser)	The core model that removes noise from the latent image
3. VAE (Autoencoder)	Compresses and decompresses images to/from latent space

Component 1: CLIP Text Encoder – Turning Language into Latents

Used to convert a **text prompt** into a vector representation (embedding) that captures the *semantic meaning* of the prompt.

- Uses **CLIP** (Contrastive Language–Image Pretraining), originally developed by OpenAI.
- CLIP is trained to **align text and image embeddings** in the same vector space.
- The idea: the prompt "a cat sitting on a table" and an actual image of a cat on a table should be *close together* in vector space.

CLIP is a is a multimodal vision and language model motivated by overcoming the fixed number of object categories when training a computer vision model. CLIP learns about images directly from raw text by jointly training on 400M (image, text) pairs. Pretraining on this scale enables zero-shot transfer to downstream tasks. CLIP pre-trains an image encoder and a text encoder to predict which images were paired with which texts in our dataset. We then use this behavior to turn CLIP into a zero-shot classifier. We convert all of a dataset's classes into captions such as "a photo of a dog" and predict the class of the caption CLIP estimates best pairs with a given image. Both features (visual and textual) are projected to a latent space with the same number of dimensions and their dot product gives a similarity score.


1. Contrastive pre-training
pepper the aussie pup
Text Encoder
Image Encoder
T_1 T_2 T_3 … T_N
I_1 I_2 I_3 ⋮ I_N
$I_1 \cdot T_1$ $I_1 \cdot T_2$ $I_1 \cdot T_3$ … $I_1 \cdot T_N$
$I_2 \cdot T_1$ $I_2 \cdot T_2$ $I_2 \cdot T_3$ … $I_2 \cdot T_N$
$I_3 \cdot T_1$ $I_3 \cdot T_2$ $I_3 \cdot T_3$ … $I_3 \cdot T_N$
$I_N \cdot T_1$ $I_N \cdot T_2$ $I_N \cdot T_3$ … $I_N \cdot T_N$
2. Create dataset classifier from label text
plane
car
dog
bird
a photo of a {object}.
Text Encoder
3. Use for zero-shot prediction
Image Encoder
I_1
$I_1 \cdot T_1$ $I_1 \cdot T_2$ $I_1 \cdot T_3$ … $I_1 \cdot T_N$
a photo of a dog.

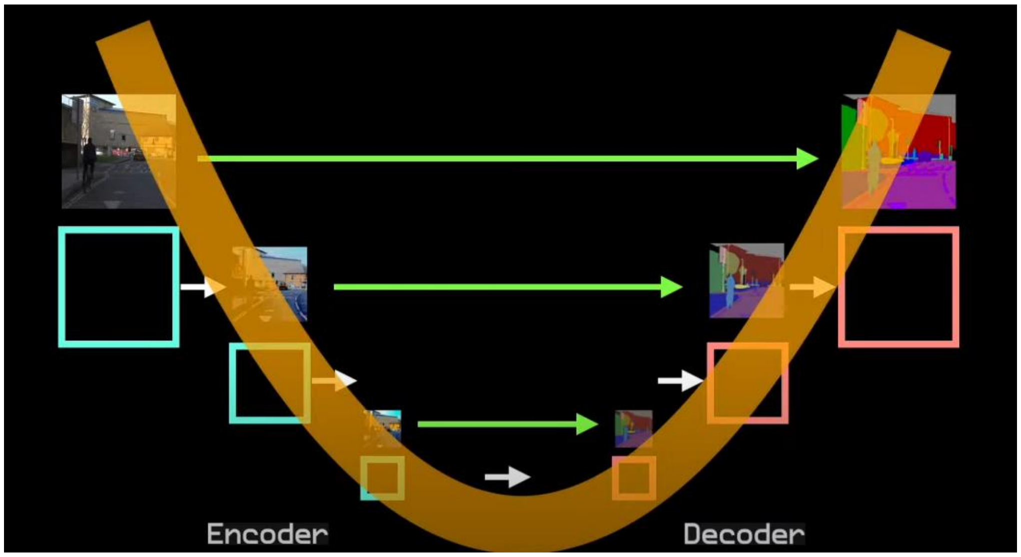
- Your input text is tokenized and passed through a Transformer-based encoder- the text encoder from CLIP is imported and used directly (OpenAI's ViT-B/32 or ViT-L/14 for SD 2.x).
- Only the text encoder is used; the image encoder is not needed because SD doesn't classify or retrieve images — it generates them.
- It takes your prompt (e.g., "a futuristic city at night") and converts it into a 768-dimensional embedding.
- This embedding is injected into the UNet denoising process as cross-attention.
- This embedding is passed into the denoising process to guide image generation.

Connection with next component (UNet): This embedding becomes a condition for the UNet during denoising — it tells the model what kind of image it should aim to produce.

Component 2: UNet (Denoising Network) – The Core of the Diffusion Process

Used to remove noise from a corrupted latent image over multiple time steps, ultimately producing a clean image latent that matches the prompt.

- Based on Denoising Diffusion Probabilistic Models (DDPMs).
- Trains to reverse a Markov chain of noise addition — from pure noise back to an image.
- Operates on latent space (compressed features), not full-resolution images.



How It Works:
During generation:

1. Start with a random noise latent z_t.
2. For each step t (e.g., from 50 down to 0):
   - UNet predicts the noise in z_t.
   - Subtract this noise to get a slightly denoised version z_{t-1}.
   - Repeat.

At each step, the CLIP text embedding conditions the UNet, helping guide it toward a meaningful output.

The UNet denoises latents, conditioned by the CLIP embedding, and prepares a final latent that can be decoded by the VAE decoder.

Component 3: Variational Autoencoder (VAE) – Bridging Pixels and Latents

To compress and decompress images between pixel space (512×512×3) and latent space (64×64×4). This makes training and inference much more efficient.

- A VAE learns a probabilistic encoding of input data.
- During training, it learns to reconstruct images via a latent distribution:

$$z = \mu + \sigma * \varepsilon, \text{ where } \varepsilon \sim N(0, 1)$$

- Encourages smooth latent space with continuous structure.

How It Works:

- Encoder: Compresses images into latent vectors (z).
- Decoder: Reconstructs images from latent vectors.
- Loss function includes:
  - Reconstruction loss (image similarity)
  - KL-divergence loss (to regularize latent distribution)

In Stable Diffusion, only the decoder is used during generation. (We don't encode existing images; we sample latent noise and decode it.)
After the UNet finishes denoising, the final latent z_0 is passed through the VAE decoder, which reconstructs the final image.

APPENDIX

VAE Encoder: A VAE encoder is the *encoding half* of a Variational Autoencoder (VAE). It compresses a high-resolution image (like 512×512×3) into a compact, latent representation (like 64×64×4).

Process:

1. Input: An image (e.g. 512×512×3)
2. It passes through multiple convolutional layers (CNNs)
3. It learns to output two things:
   - A mean vector (μ) and
   - A standard deviation vector (σ)
4. From these, a latent vector z is sampled:

   $$z = \mu + \sigma * \varepsilon, \text{ where } \varepsilon \sim N(0, 1)$$

5. This sampled z is the latent code: a compressed, meaningful version of the input image.
   This “sampling” step is what makes it variational — it adds randomness, allowing the model to generalize better and learn a smooth latent space.

Multimodal: In AI, multimodal refers to models that can understand and process multiple types of input data—called *modalities*—at the same time or in combination.

Zero-shot: The model can perform a task without having been explicitly trained on that task. It generalizes using what it has learned from pretraining. Specifically, a zero-shot classifier is an AI model that can classify data into categories it has never seen during training—without any labeled examples from those categories.

768-dimensional embedding: A 768-dimensional embedding is a vector with 768 numerical values (i.e., a 1×768 array) that represents the semantic meaning of your text prompt in a way that the model can understand. Each of these 768 numbers represents a different “semantic direction” in that space — some might correspond (abstractly) to "darkness", "buildings", "modern", etc. The number '768' is not magical — it comes from the architecture of the model, and specifically from: The Size of the Last Hidden Layer of the CLIP Text Encoder (Stable Diffusion 1.4/1.5 uses OpenAI's CLIP ViT-B/32 model, which has the output embedding size (i.e., the dimension of the final transformer layer) set to 768. It’s simply the width of the vector space the model was trained to use. A higher-dimensional space gives more representational capacity. This is Useful in Stable Diffusion because Stable Diffusion’s UNet needs a rich, continuous input that describes the prompt.

Cross-attention: Attention allows a model to focus on specific parts of an input when generating an output — like focusing on the word "dog" when drawing a dog. Cross-attention is a mechanism where the model attends to a different modality or sequence — such as text features attending to image features. In Stable Diffusion, this means: The image generation process (in UNet) is guided by your text prompt embedding and Each image feature (pixel or latent patch) can look at the text and ask: “Which words are relevant to this part of the image?”