Controllable Generation

Lecturer: Qiang Sun Lecture 06

1 Building an Image Generator

In the previous sections, we learned how to train a flow matching or diffusion model to sample from a distribution $p_{\text{data}}(x)$. This recipe is general and can be applied to a variety of different data types and applications. In this section, we learn how to apply this framework to build an image or video generator, such as e.g., Stable Diffusion 3 and Meta Movie Gen Video. To build such a model, there are two main ingredients that we are missing:

- We will need to formulate **conditional generation** (**guidance**), e.g., how do we generate an image that fits a specific text prompt, and how our existing objectives may be suitably adapted to this end. We will also learn about **classifier-free guidance**, a popular technique used to enhance the quality of conditional generation.
- We will discuss common neural network architectures, focusing on those designed for images and videos.

Finally, we will examine in depth the two state-of-the-art image and video models mentioned above, *Stable Diffusion* and *Meta MovieGen*, to give you a taste of how things are done at scale.

1.1 Guidance

So far, the generative models we considered were **unconditional**, e.g. an image model would simply generate some image. However, the task is not merely to generate an arbitrary object, but to generate an object **conditioned on some additional information**. For example, one might imagine a generative model for images which takes in a text prompt y, and then generates an image x conditioned on y. For fixed prompt y, we would thus like to sample from $p_{\text{data}}(x|y)$, that is, the data distribution conditioned on y.

Formally, we think of y to live in a space $\mathcal{Y}$. When y corresponds to a text prompt, for example, $\mathcal{Y}$ would likely be some continuous space like $\mathbb{R}^{d_y}$. When y corresponds to some discrete class label, $\mathcal{Y}$ would be discrete. In the lab, we will work with the MNIST dataset, in which case we will take $\mathcal{Y} = \{0, 1, \ldots, 9\}$ to correspond to the identities of handwritten digits.

To avoid a notation and terminology clash with the use of the word "conditional" to refer to conditioning on $z \sim p_{\text{data}}$ (conditional probability path / vector field), we will make use of the term *guided* to refer specifically to conditioning on y.

Remark 1 (Guided vs. Conditional Terminology)

In these notes, we opt to use the term *guided* in place of *conditional* to refer to the act of conditioning on y. Here, we will refer to e.g. a guided vector field $u_t^{\text{target}}(x|y)$ and a conditional vector field $u_t^{\text{target}}(x|z)$. This terminology is consistent with other works such as [15].

The goal of guided generative modeling is thus to be able to sample from $p_{\text{data}}(x|y)$ for any such y.

Key Idea 1 (Guided Generative Model)

We define a guided diffusion model to consist of a guided vector field $u_t^\theta(\cdot|y)$, parameterized by some neural network, and a time-dependent diffusion coefficient σ_t, together given by

$$\begin{aligned} \textbf{Neural network:} \quad & u^\theta : \mathbb{R}^d \times \mathcal{Y} \times [0,1] \to \mathbb{R}^d, \qquad (x, y, t) \mapsto u_t^\theta(x|y), \\ \textbf{Fixed:} \quad & \sigma_t : [0,1] \to [0,1), \qquad t \mapsto \sigma_t. \end{aligned}$$

Notice the difference from the unguided case: we are additionally guiding u_t^θ with the input $y \in \mathcal{Y}$.

For any such y, samples may then be generated from such a model as follows:

Initialization: $X_0 \sim p_{\text{init}}$. ▶ Initialize with a simple distribution (such as a Gaussian).

Simulation: $\mathrm{d}X_t = u_t^\theta(X_t|y)\,\mathrm{d}t + \sigma_t\,\mathrm{d}W_t$. ▶ Simulate SDE from $t = 0$ to $t = 1$.

Goal: $X_1 \sim p_{\text{data}}(\cdot|y)$. ▶ Goal is for X_1 to be distributed like $p_{\text{data}}(\cdot|y)$.

When $\sigma_t = 0$, we say that such a model is a **guided flow model**.

1.1.1 Guidance for Flow Models

If we imagine fixing our choice of y, and take our data distribution as $p_{\text{data}}(x|y)$, then we have recovered the unguided generative problem, and can accordingly construct a generative model using the conditional flow matching objective, viz.,

$$\mathbb{E}_{z \sim p_{\text{data}}(\cdot|y),\, x \sim p_t(\cdot|z)} \left\| u_t^\theta(x|y) - u_t^{\text{target}}(x|z) \right\|^2. \tag{1.1}$$

Note that the label y does not affect the conditional probability path $p_t(\cdot|z)$ or the conditional vector field $u_t^{\text{target}}(x|z)$ (although, in principle, we could make it dependent). Expanding the expectation over all such choices of y, and over all times $t \sim \text{Unif}[0,1)$, we thus obtain a **guided conditional flow matching objective**

$$\mathcal{L}_{\text{CFM}}^{\text{guided}}(\theta) = \mathbb{E}_{(z,y) \sim p_{\text{data}}(z,y),\, t \sim \text{Unif}[0,1),\, x \sim p_t(\cdot|z)} \left\| u_t^\theta(x|y) - u_t^{\text{target}}(x|z) \right\|^2. \tag{1.2}$$

One of the main differences between the guided objective in eq. (1.2) and the unguided objective is that here we are sampling $(z, y) \sim p_{\text{data}}$ rather than just $z \sim p_{\text{data}}$. The reason

is that our data distribution is now, in principle, a joint distribution over e.g. both images z and text prompts y. In practice, this means that a PyTorch implementation of eq. (1.2) would involve a dataloader which returned batches of both z and y. The above procedure leads to a faithful generation procedure of $p_{\text{data}}(\cdot|y)$.

Classifier-free guidance. While the above conditional training procedure is theoretically valid, it was soon empirically realized that image samples with this procedure did not fit well enough to the desired label y. It was discovered that perceptual quality is increased when the effect of the guidance variable y is artificially reinforced. This insight was distilled into a technique known as **classifier-free guidance** that is widely used in the context of state-of-the-art diffusion models.

For simplicity, we will focus here on the case of Gaussian probability paths. Recall from previous sections that a Gaussian conditional probability path is given by

$$p_t(\cdot|z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d),$$

where the noise schedulers α_t and β_t are continuously differentiable, monotonic, and satisfy $\alpha_0 = \beta_1 = 0$ and $\alpha_1 = \beta_0 = 1$.

To gain intuition for classifier-free guidance, we recall the following proposition from Lecture 5.

Proposition 1 (Conversion formula for Gaussian probability paths)

For the Gaussian probability path $p_t(\cdot|z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$, it holds that the conditional (resp. marginal) vector field can be converted into the conditional (resp. marginal) score via

$$u_t^{\text{target}}(x|z) = \left(\frac{\beta_t^2 \dot{\alpha}_t}{\alpha_t} - \beta_t \dot{\beta}_t \right) \nabla \log p_t(x|z) + \frac{\dot{\alpha}_t}{\alpha_t} x,$$
$$u_t^{\text{target}}(x) = \left(\frac{\beta_t^2 \dot{\alpha}_t}{\alpha_t} - \beta_t \dot{\beta}_t \right) \nabla \log p_t(x) + \frac{\dot{\alpha}_t}{\alpha_t} x.$$

Here the formula for the above marginal vector field u_t^{target} is called the **probability flow ODE** in the literature (more precisely, the corresponding ODE).

Using the above proposition, we can rewrite the guided vector field $u_t^{\text{target}}(x|y)$ in the following form using the guided score function $\nabla \log p_t(x|y)$:

$$u_t^{\text{target}}(x|y) = a_t x + b_t \nabla \log p_t(x|y), \tag{1.3}$$

where

$$(a_t, b_t) = \left(\frac{\dot{\alpha}_t}{\alpha_t}, \frac{\dot{\alpha}_t \beta_t^2 - \dot{\beta}_t \beta_t \alpha_t}{\alpha_t} \right). \tag{1.4}$$

However, notice that by Bayes' rule, we can rewrite the guided score as

$$\nabla \log p_t(x|y) = \nabla \log \left(\frac{p_t(x)\, p_t(y|x)}{p_t(y)} \right) = \nabla \log p_t(x) + \nabla \log p_t(y|x), \tag{1.5}$$

where we used that the gradient ∇ is taken with respect to the variable x, so that $\nabla \log p_t(y) = 0$. We may thus rewrite

$$u_t^{\text{target}}(x|y) = a_t x + b_t\big(\nabla \log p_t(x) + \nabla \log p_t(y|x)\big) = u_t^{\text{target}}(x) + b_t \nabla \log p_t(y|x).$$

Notice the shape of the above equation: the guided vector field $u_t^{\text{target}}(x|y)$ is a sum of the unguided vector field plus a guided "classifier" term $\nabla \log p_t(y|x)$. Since people observed that their image x did not fit their prompt y well enough, it was a natural idea to scale up the contribution of the $\nabla \log p_t(y|x)$ term, yielding

$$\widetilde{u}_t(x|y) = u_t^{\text{target}}(x) + w\, b_t \nabla \log p_t(y|x),$$

where $w > 1$ is known as the **guidance scale**. Note that this is a heuristic: for $w \neq 1$, it holds that $\widetilde{u}_t(x|y) \neq u_t^{\text{target}}(x|y)$, i.e. it is not the true guided vector field. However, empirical results have shown this to yield preferable results (when $w > 1$).

Remark 2 (Where is the classifier?)

The term $\log p_t(y|x)$ can be considered as a sort of classifier of noised data (i.e. it gives the likelihood of y given x). In fact, early works in diffusion trained actual classifiers and used them to guide via the above procedure. This leads to **classifier guidance** (Dhariwal and Nichol, 2021; Song et al., 2020). As it has been largely superseded by classifier-free guidance, we do not consider it here.

We may again apply the equality $\nabla \log p_t(x|y) = \nabla \log p_t(x) + \nabla \log p_t(y|x)$ to obtain

$$\begin{aligned}
\widetilde{u}_t(x|y) &= u_t^{\text{target}}(x) + w b_t \nabla \log p_t(y|x) \\
&= u_t^{\text{target}}(x) + w b_t\big(\nabla \log p_t(x|y) - \nabla \log p_t(x)\big) \\
&= (1-w) u_t^{\text{target}}(x) + w\, u_t^{\text{target}}(x|y).
\end{aligned}$$

We may therefore express the scaled guided vector field $\widetilde{u}_t(x|y)$ as a linear combination of the unguided vector field $u_t^{\text{target}}(x)$ with the guided vector field $u_t^{\text{target}}(x|y)$.

The idea might then be to train both an unguided $u_t^{\text{target}}(x)$ (using e.g. the CMF objective) as well as a guided $u_t^{\text{target}}(x|y)$ (using e.g. eq. (1.2)), and then combine them at inference time to obtain $\widetilde{u}_t(x|y)$. "But wait!" you might ask, "wouldn't we need to train two models then?" It turns out we can train both in one model: we may augment our label set with a new, additional $\varnothing$ label that denotes the absence of conditioning. We can then treat $u_t^{\text{target}}(x) = u_t^{\text{target}}(x|\varnothing)$. With that, we do not need to train a separate model to reinforce the effect of a hypothetical classifier. This approach of training a conditional and unconditional model in one (and subsequently reinforcing the conditioning) is known as *classifier-free guidance* (CFG) (Ho and Salimans, 2022).

Remark 3 (Derivation for general probability paths)

Note that the construction

$$\widetilde{u}_t(x|y) = (1-w) u_t^{\text{target}}(x) + w\, u_t^{\text{target}}(x|y)$$

is equally valid for any choice of probability path, not just a Gaussian one. When $w = 1$, it is straightforward to verify that $\widetilde{u}_t(x|y) = u_t^{\text{target}}(x|y)$. Our derivation using Gaussian paths was simply to illustrate the intuition behind the construction, and in particular of amplifying the contribution of a "classifier" term $\nabla \log p_t(y|x)$.

Training and context-free guidance. We must now amend the guided conditional flow matching objective eq. (64) to account for the possibility of $y = \varnothing$. The challenge is that when sampling $(z, y) \sim p_{\text{data}}$, we will never obtain $y = \varnothing$. It follows that we must introduce the possibility of $y = \varnothing$ artificially. To do so, we define a hyperparameter η to be the probability that we discard the original label y and replace it with $\varnothing$. We thus arrive at our **CFG conditional flow matching training objective**

$$\mathcal{L}_{\text{CFM}}^{\text{CFG}}(\theta) = \mathbb{E}_{\square} \left\| u_t^\theta(x|y) - u_t^{\text{target}}(x|z) \right\|^2, \tag{1.6}$$

where

$$\square : \ (z, y) \sim p_{\text{data}}(z, y), \quad t \sim \text{Unif}[0, 1), \quad x \sim p_t(\cdot|z), \quad \text{replace } y = \varnothing \text{ with prob. } \eta. \tag{1.7}$$

Summary 1 (Classifier-Free Guidance for Flow Models)

Given the unguided marginal vector field $u_t^{\text{target}}(x|\varnothing)$, the guided marginal vector field $u_t^{\text{target}}(x|y)$, and a **guidance scale** $w > 1$, we define **the classifier-free guided vector field** $\widetilde{u}_t(x|y)$ by

$$\widetilde{u}_t(x|y) = (1 - w)\, u_t^{\text{target}}(x|\varnothing) + w\, u_t^{\text{target}}(x|y). \tag{1.8}$$

By approximating $u_t^{\text{target}}(x|\varnothing)$ and $u_t^{\text{target}}(x|y)$ using the same neural network, we may leverage the following **classifier-free guidance CFM** (CFG-CFM) objective:

$$\mathcal{L}_{\text{CFM}}^{\text{CFG}}(\theta) = \mathbb{E}_{\square} \left\| u_t^\theta(x|y) - u_t^{\text{target}}(x|z) \right\|^2, \tag{1.9}$$

where

$$\square : \ (z, y) \sim p_{\text{data}}(z, y), \quad t \sim \text{Unif}[0, 1), \quad x \sim p_t(\cdot|z), \quad \text{replace } y = \varnothing \text{ with prob. } \eta. \tag{1.10}$$

In plain English, $\mathcal{L}_{\text{CFM}}^{\text{CFG}}$ might be approximated by:

$(z, y) \sim p_{\text{data}}(z, y)$	▶ Sample (z, y) from the data distribution.
$t \sim \text{Unif}[0, 1)$	▶ Sample t uniformly on $[0, 1)$.
$x \sim p_t(\cdot\|z)$	▶ Sample x from the conditional prob. path $p_t(x\|z)$.
With prob. η, replace $y \leftarrow \varnothing$	▶ Drop the label.
$\mathcal{L}_{\text{CFM}}^{\text{CFG}}(\theta) = \|u_t^\theta(x\|y) - u_t^{\text{target}}(x\|z)\|^2$	▶ Regress against the conditional vector field.

Above, we made use multiple times of the fact that $u_t^{\text{target}}(x|z) = u_t^{\text{target}}(x|z, y)$. At inference time, for a fixed choice of y, we may sample via:

Figure 1: The effect of classifier guidance. The prompt here is the class chosen to be "Corgi" (a specific type of dog). Left: samples generated with no guidance (i.e., $w = 1$). Right: samples generated with classifier guidance and $w = 4$. As shown, classifier-free guidance improves the similarity to the prompt. Figure taken from Ho and Salimans (2022).

Algorithm 1: Classifier-free guidance training for Gaussian probability path $p_t(x|z) = \mathcal{N}(x; \alpha_t z, \beta_t^2 I_d)$

Require: Paired dataset $(z, y) \sim p_{\text{data}}$, neural network u_t^θ
for *each mini-batch of data* **do**
  Sample a data example (z, y) from the dataset;
  Sample a random time $t \sim \text{Unif}[0, 1]$;
  Sample noise $\varepsilon \sim \mathcal{N}(0, I_d)$;
  Set $x = \alpha_t z + \beta_t \varepsilon$;
  With probability p drop label: $y \leftarrow \varnothing$;
  Compute loss $\mathcal{L}(\theta) = \left\| u_t^\theta(x|y) - (\dot{\alpha}_t \varepsilon + \dot{\beta}_t z) \right\|^2$;
  Update the model parameters θ via gradient descent on $\mathcal{L}(\theta)$;

Initialization: $X_0 \sim p_{\text{init}}(x)$ ▶ Initialize with a simple distribution (such as a Gaussian).

Simulation: $\mathrm{d}X_t = \widetilde{u}_t^\theta(X_t|y)\,\mathrm{d}t$ ▶ Simulate ODE from $t = 0$ to $t = 1$.

Samples: X_1 ▶ Goal is for X_1 to adhere to the guiding variable y.

Note that the distribution of X_1 is not necessarily aligned with $X_1 \sim p_{\text{data}}(\cdot|y)$ anymore if we use a weight $w > 1$. However, empirically, this shows better alignment with conditioning.

In Fig. 1, we illustrated class-based classifier-free guidance on 128×128 ImageNet, as in Ho and Salimans (2022). Similarly, in Fig. 2 we visualize the effect of various guidance scales w when applying classifier-free guidance to sampling from the MNIST dataset of handwritten digits.

Figure 2: The effect of classifier-free guidance applied at various guidance scales for the MNIST dataset of handwritten digits. Left: guidance scale set to $w = 1.0$. Middle: guidance scale set to $w = 2.0$. Right: guidance scale set to $w = 4.0$. You will generate a similar image yourself in lab three!

1.1.2 Guidance for Diffusion Models

We now extend the reasoning of the previous section to diffusion models. First, in the same way that we obtained eq. (1.2) we may generalize the conditional score matching loss to obtain the **guided conditional score matching objective**

$$\mathcal{L}_{\text{CSM}}^{\text{guided}}(\theta) = \mathbb{E}_{\square}\big[\|s_t^\theta(x|y) - \nabla \log p_t(x|z)\|^2\big], \tag{1.11}$$

where

$$\square: \ (z, y) \sim p_{\text{data}}(z, y), \quad t \sim \text{Unif}, \quad x \sim p_t(\cdot|z). \tag{1.12}$$

A guided score network $s_t^\theta(x|y)$ trained with eq. (1.11) might then be combined with the guided vector field $u_t^\theta(x|y)$ to simulate the SDE

$$X_0 \sim p_{\text{init}}, \qquad \mathrm{d}X_t = \Big[u_t^\theta(X_t|y) + \frac{\sigma_t^2}{2} s_t^\theta(X_t|y)\Big]\mathrm{d}t + \sigma_t \, \mathrm{d}W_t.$$

Classifier-free guidance. We now extend classifier-free guidance to the diffusion setting. By Bayes' rule (see eq. (1.5)), we have

$$\nabla \log p_t(x|y) = \nabla \log p_t(x) + \nabla \log p_t(y|x),$$

so that for guidance scale $w > 1$ we may define

$$\begin{aligned}
\widetilde{s}_t(x|y) &= \nabla \log p_t(x) + w \nabla \log p_t(y|x) \\
&= \nabla \log p_t(x) + w\big(\nabla \log p_t(x|y) - \nabla \log p_t(x)\big) \\
&= (1-w)\nabla \log p_t(x) + w \nabla \log p_t(x|y) \\
&= (1-w)\nabla \log p_t(x|\varnothing) + w \nabla \log p_t(x|y).
\end{aligned}$$

We thus arrive at the CFG-compatible (that is, accounting for the possibility of $\varnothing$) objective

$$\mathcal{L}_{\text{DSM}}^{\text{CFG}}(\theta) = \mathbb{E}_{\square}\big\|s_t^\theta(x|y) - \nabla \log p_t(x|z)\big\|^2, \tag{1.13}$$

where

$$\square: \ (z, y) \sim p_{\text{data}}(z, y), \quad t \sim \text{Unif}[0, 1), \quad x \sim p_t(\cdot|z), \quad \text{replace } y = \varnothing \text{ with prob. } \eta, \tag{1.14}$$

where η is a hyperparameter (the probability of replacing y with $\varnothing$). We will refer to the above objective $\mathcal{L}_{\text{DSM}}^{\text{CFG}}(\theta)$ as the **classifier-free guidance CSM** (CFG-CSM) objective. We recap as follows.

Summary 2 (Classifier-Free Guidance for Diffusions)

Given the unguided marginal score $\nabla \log p_t(x|\varnothing)$, the guided marginal score field $\nabla \log p_t(x|y)$, and a **guidance scale** $w > 1$, we define the **classifier-free guided score** $\widetilde{s}_t(x|y)$ by

$$\widetilde{s}_t(x|y) = (1 - w)\nabla \log p_t(x|\varnothing) + w\nabla \log p_t(x|y). \tag{1.15}$$

By approximating $\nabla \log p_t(x|\varnothing)$ and $\nabla \log p_t(x|y)$ using the same neural network $s_t^\theta(x|y)$, we may leverage the following **classifier-free guidance CSM** (CFG-CSM) objective:

$$\mathcal{L}_{\text{CSM}}^{\text{CFG}}(\theta) = \mathbb{E}_{\square} \left\| s_t^\theta(x|(1 - \xi)y + \xi\varnothing) - \nabla \log p_t(x|z) \right\|^2, \tag{1.16}$$

where

$$\square: \ (z, y) \sim p_{\text{data}}(z, y), \quad t \sim \text{Unif}[0, 1), \quad x \sim p_t(\cdot|z), \quad \text{replace } y = \varnothing \text{ with prob. } \eta. \tag{1.17}$$

(Here, $\xi \in \{0, 1\}$ can be viewed as the Bernoulli indicator of whether we dropped the label.)

In plain English, $\mathcal{L}_{\text{DSM}}^{\text{CFG}}$ might be approximated by:

$(z, y) \sim p_{\text{data}}(z, y)$	▶ Sample (z, y) from the data distribution.
$t \sim \text{Unif}[0, 1)$	▶ Sample t uniformly on $[0, 1)$.
$x \sim p_t(\cdot\|z)$	▶ Sample x from the conditional probability path $p_t(x\|z)$.
With prob. η, replace $y \leftarrow \varnothing$	▶ Drop the label.
$\mathcal{L}_{\text{DSM}}^{\text{CFG}}(\theta) = \|s_t^\theta(x\|y) - \nabla \log p_t(x\|z)\|^2$	▶ Regress against the conditional score.

At inference time, for a fixed choice of $w > 1$, we may combine $s_t^\theta(x|y)$ with a guided vector field $u_t^\theta(x|y)$ and define

$$\widetilde{s}_t^\theta(x|y) = (1 - w)s_t^\theta(x|\varnothing) + ws_t^\theta(x|y),$$
$$\widetilde{u}_t^\theta(x|y) = (1 - w)u_t^\theta(x|\varnothing) + wu_t^\theta(x|y).$$

Then we may sample via:

Initialization: $X_0 \sim p_{\text{init}}(x)$ ▶ Initialize with a simple distribution (such as a Gaussian).

Simulation: $\mathrm{d}X_t = \left[\widetilde{u}_t^\theta(X_t|y) + \frac{\sigma_t^2}{2}\widetilde{s}_t^\theta(X_t|y)\right]\mathrm{d}t + \sigma_t \, \mathrm{d}W_t$ ▶ Simulate SDE from $t = 0$ to $t = 1$.

Samples: X_1 ▶ Goal is for X_1 to adhere to the guiding variable y.

1.2 Neural network architectures

We next discuss the design of neural networks for flow and diffusion models. Specifically, we answer the question of how to construct a neural network architecture that represents the (guided) vector field $u_t^\theta(x|y)$ with parameters θ.

Note that the neural network must have three inputs: a vector $x \in \mathbb{R}^d$, a conditioning variable $y \in \mathcal{Y}$, and a time value $t \in [0, 1]$; and one output: a vector $u_t^\theta(x|y) \in \mathbb{R}^d$.

For low-dimensional distributions (e.g. the toy distributions we have seen in previous sections), it is sufficient to parameterize $u_t^\theta(x|y)$ as a multi-layer perceptron (MLP), otherwise known as a fully connected neural network. That is, in this simple setting, a forward pass through $u_t^\theta(x|y)$ would involve concatenating our input x, y, t and passing them through an MLP.

However, for complex, high-dimensional distributions, such as those over images, videos, and proteins, an MLP is rarely sufficient, and it is common to use special, application-specific architectures. For the remainder of this section, we will consider the case of images (and by extension, videos), and discuss two common architectures: the **U-Net** (Ronneberger et al., 2015) and the **diffusion transformer** (DiT).

1.2.1 U-Nets and Diffusion Transformers

Before we dive into the specifics of these architectures, recall that an image is simply a vector $x \in \mathbb{R}^{C_{\text{image}} \times H \times W}$. Here C_{image} denotes the number of **channels** (an RGB image typically would have $C_{\text{image}} = 3$ color channels), H denotes the **height** of the image in pixels, and W denotes the **width** of the image in pixels.

U-Nets. The **U-Net** architecture (Ronneberger et al., 2015) is a specific type of convolutional neural network. Originally designed for image segmentation, its crucial feature is that both its input and its output have the shape of images (possibly with a different number of channels). This makes it ideal to parameterize a vector field $x \mapsto u_t^\theta(x|y)$: for fixed y, t its input has the shape of an image and its output does, too. Therefore, U-Nets were widely used in the development of diffusion models.

A U-Net consists of a series of **encoders** E_i, a corresponding sequence of **decoders** D_i, and a latent processing block in between which we refer to as a **midcoder** ("midcoder" is not a standard term in the literature). For sake of example, let us walk through the path taken by an image $x_t \in \mathbb{R}^{3 \times 256 \times 256}$ as it is processed by the U-Net:

$$x_t^{\text{input}} \in \mathbb{R}^{3 \times 256 \times 256} \qquad \blacktriangleright \text{ Input to the U-Net.}$$

$$x_t^{\text{latent}} = E(x_t^{\text{input}}) \in \mathbb{R}^{512 \times 32 \times 32} \qquad \blacktriangleright \text{ Pass through encoders to obtain latent.}$$

$$x_t^{\text{latent}} = M(x_t^{\text{latent}}) \in \mathbb{R}^{512 \times 32 \times 32} \qquad \blacktriangleright \text{ Pass latent through midcoder.}$$

$$x_t^{\text{output}} = D(x_t^{\text{latent}}) \in \mathbb{R}^{3 \times 256 \times 256} \qquad \blacktriangleright \text{ Pass through decoders to obtain output.}$$

Notice that as the input passes through the encoders, the number of channels in its representation increases, while the height and width of the images are decreased. Both the encoder and the decoder usually consist of a series of convolutional layers (with activation functions,

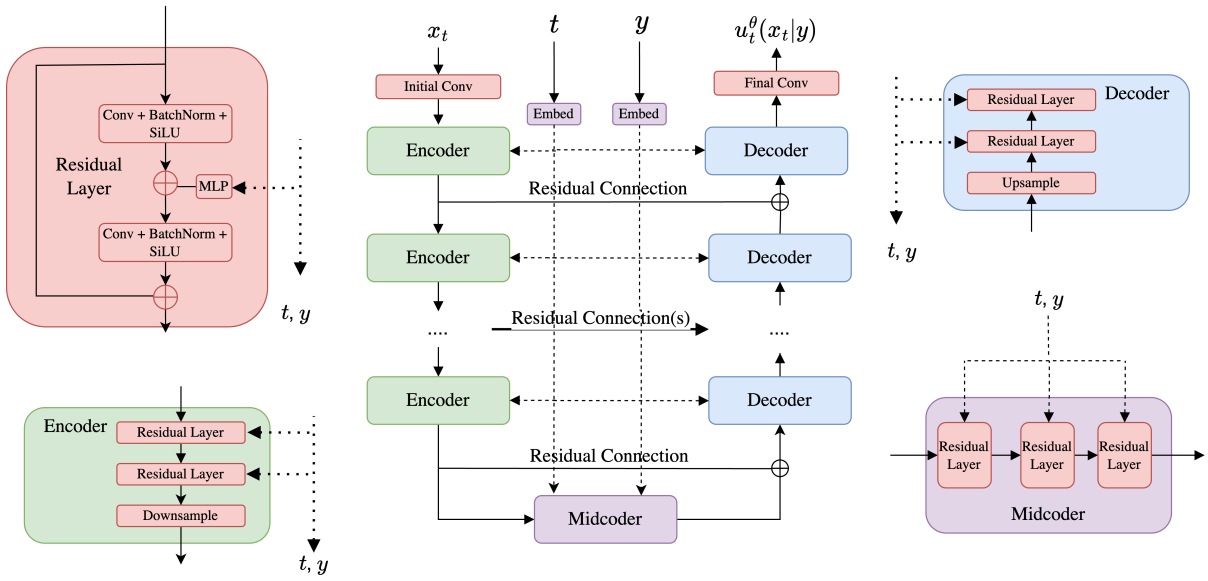

Figure 3: The simplified U-Net architecture used in the homework.

pooling operations, etc. in between). Not shown above are two points: (i) the input is often fed into an initial pre-encoding block to increase the number of channels before being fed into the first encoder block; and (ii) the encoders and decoders are often connected by **residual connections**. The complete picture is shown in Fig. 3. At a high level, most U-Nets involve some variant of what is described above. However, certain design choices may differ in practice; in particular, it is common to include attention layers throughout the encoders and decoders. The U-Net derives its name from the "U"-like shape formed by its encoders and decoders.

Diffusion transformers. One alternative to U-Nets are **diffusion transformers** (DiTs), which dispense with convolutions and purely use **attention** (Vaswani et al., 2017; Peebles and Xie, 2023). Diffusion transformers are based on **vision transformers** (ViTs), in which the big idea is essentially to divide up an image into patches, embed each of these patches, and then attend between the patches (Dosovitskiy et al., 2020). *Stable Diffusion 3*, trained with conditional flow matching, parameterizes the velocity field $u_t^\theta(x)$ as a modified DiT, as we discuss later (Esser et al., 2024).

Remark 4 (Working in Latent Space)

A common problem for large-scale applications is that the data is so high-dimensional that it consumes too much memory. For example, we might want to generate a high resolution image of 1000×10000 pixels, leading to 1 million dimensions. To reduce memory usage, a common design pattern is to work in a **latent space** that can be considered a compressed version of our data at lower resolution.

Specifically, the usual approach is to combine a flow or diffusion model with a (variational) **autoencoder** (Rombach et al., 2022). In this case, one first encodes the training dataset in the latent space via an autoencoder, and then trains the flow or diffusion model in the latent space.

Sampling is performed by first sampling in the latent space using the trained flow or diffusion model, and then decoding the output via the decoder. Intuitively, a well- trained autoencoder can be thought of as filtering out semantically meaningless details, allowing the generative model to "focus" on important, perceptually relevant features (Vahdat et al., 2021). By now, nearly all state-of-the-art approaches to image and video generation involve training a flow or diffusion model in the latent space of an autoencoder – so-called **latent diffusion models** (Rombach et al., 2022; Vahdat et al., 2021). However, it is important to note: one also needs to train the autoencoder before training the diffusion model. Crucially, performance now depends also on how good the autoencoder compresses images into latent space and recovers aesthetically pleasing images.

1.2.2 Encoding the Guiding Variable

Up until this point, we have glossed over how exactly the guiding (conditioning) variable y is fed into the neural network $u_t^\theta(x|y)$. Broadly, this process can be decomposed into two steps:

- embedding the raw input y_{raw} (e.g., the text prompt "a cat playing a trumpet, photorealistic") into some vector-valued input y, and
- feeding the resulting embedding y into the actual model.

Embedding raw input. We consider two cases: (1) where y_{raw} is a discrete class label, and (2) where y_{raw} is a text prompt.

When $y_{\text{raw}} \in \{0, \dots, N\}$ is just a class label, then it is often easiest to simply learn a separate embedding vector for each of the $N + 1$ possible values of y_{raw}, and set y to this embedding vector. One would consider the parameters of these embeddings to be included in the parameters of $u_t^\theta(x|y)$, and would therefore learn these during training.

When y_{raw} is a text prompt, the situation is more complex, and approaches largely rely on frozen, pre-trained models. Such models are trained to embed a discrete text input into a continuous vector that captures the relevant information.

One such model is known as CLIP (Contrastive Language–Image Pre-training). CLIP is trained to learn a shared embedding space for both images and text prompts, using a training loss designed to encourage image embeddings to be close to their corresponding prompts, while being farther from the embeddings of other images and prompts (Radford et al., 2021). We might therefore take $y = \text{CLIP}(y_{\text{raw}}) \in \mathbb{R}^{d_{\text{CLIP}}}$ to be the embedding produced by a frozen, pre-trained CLIP model.

In certain cases, it may be undesirable to compress the entire text sequence into a single representation. In this case, one might additionally consider embedding the prompt using a pre-trained transformer so as to obtain a sequence of embeddings. It is also common to combine multiple such pretrained embeddings when conditioning so as to simultaneously reap the benefits of each model (Polyak et al., 2024).

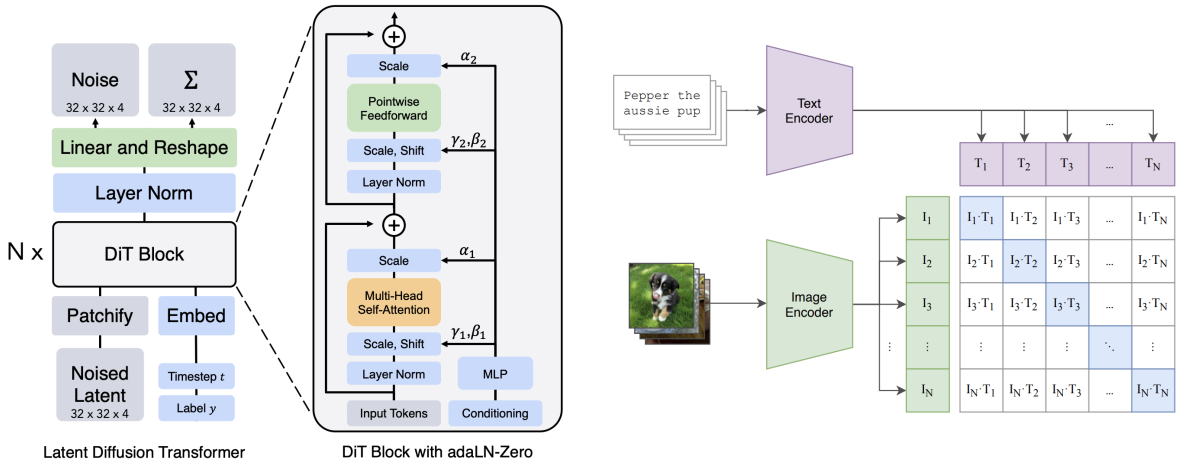

Figure 4: Left: an overview of the diffusion transformer architecture, taken from (Esser et al., 2024). Right: a schematic of the contrastive CLIP loss, in which a shared image-text embedding space is learned, taken from (Radford et al., 2021).

Feeding in the embedding. Suppose now that we have obtained our embedding vector $y \in \mathbb{R}^{d_y}$. The answer varies, but usually it is some variant of the following: feed it individually into every sub-component of the architecture.

Let us briefly describe how this is accomplished in the U-Net implementation used in lab three (Fig. 13). At some intermediate point within the network, we would like to inject information from $y \in \mathbb{R}^{d_y}$ into the current activation $x_t^{\text{intermediate}} \in \mathbb{R}^{C \times H \times W}$. We might do so using the following PyTorch-esque pseudocode:

$$y \leftarrow \text{MLP}(y) \in \mathbb{R}^C \qquad \blacktriangleright \text{ Map } y \text{ from } \mathbb{R}^{d_y} \text{ to } \mathbb{R}^C.$$

$$y \leftarrow \text{reshape}(y) \in \mathbb{R}^{C \times 1 \times 1} \qquad \blacktriangleright \text{ Reshape } y \text{ to "look" like an image.}$$

$$x_t^{\text{intermediate}} \leftarrow \text{broadcast\_add}(x_t^{\text{intermediate}}, y) \in \mathbb{R}^{C \times H \times W} \qquad \blacktriangleright \text{ Add } y \text{ to } x_t^{\text{intermediate}} \text{ pointwise.}$$

One exception to this simple pointwise conditioning scheme is when we have a *sequence* of embeddings as produced by some pretrained language model. In this case, we might consider using some sort of cross-attention scheme between our image (suitably patchified) and the tokens of the embedded sequence. We will see multiple examples of this later.

1.3 A Survey of Large-Scale Image and Video Models

We conclude this section by briefly examining two large-scale generative models: Stable Diffusion 3 for image generation and Meta's Movie Gen Video for video generation (Polyak et al., 2024). As you will see, these models use the techniques we have described in this work along with additional architectural enhancements to both scale and accommodate richly structured conditioning modalities, such as text-based input.

1.3.1 Stable Diffusion 3

Stable Diffusion is a series of state-of-the-art image generation models. These models were among the first to use large-scale latent diffusion models for image generation. If you have not done so, we recommend testing it for yourself online: `https://stability.ai/news/stable-diffusion-3`.

Stable Diffusion 3 uses the same conditional flow matching objective that we study in this work (see Algorithm 1).[1] As outlined in their paper, they extensively tested various flow and diffusion alternatives and found flow matching to perform best. For training, it uses classifier-free guidance training (with dropping class labels) as outlined above. Further, Stable Diffusion 3 follows the approach outlined in Section 5.2 by training within the latent space of a pre-trained autoencoder. Training a good autoencoder was a big contribution of the first stable diffusion papers.

To enhance text conditioning, Stable Diffusion 3 uses three different types of text embeddings (including CLIP embeddings as well as the sequential outputs produced by a pretrained instance of the encoder of Google's T5-XXL (Raffel et al., 2020), and similar to approaches taken in (Esser et al., 2024; Saharia et al., 2022)). Whereas CLIP embeddings provide a coarse, overarching embedding of the input text, the T5 embeddings provide a more granular level of context, allowing for the possibility of the model attending to particular elements of the conditioning text.

To accommodate these sequential context embeddings, the authors propose to extend the diffusion transformer to attend not just to patches of the image, but to the text embeddings as well, thereby extending the conditioning capacity from the class-based scheme originally proposed for DiT to sequential context embeddings. This proposed modified DiT is referred to as a *multi-modal DiT* (MM-DiT), and is depicted in Fig. 5. Their final, largest model has **8 billion parameters**.

For sampling, they use 50 steps (i.e. they have to evaluate the network 50 times) using an Euler simulation scheme and a classifier-free guidance weight between 2.0–5.0.

1.3.2 Meta Movie Gen Video

Next, we discuss Meta's video generator, Movie Gen Video: `https://ai.meta.com/research/movie-gen/`.

As the data are not images but videos, the data x lie in the space $\mathbb{R}^{T\times C\times H\times W}$ where T represents the temporal dimension (i.e. the number of frames). Many of the design choices made in this video setting can be seen as adapting existing techniques (e.g., autoencoders, diffusion transformers, etc.) from the image setting to handle this extra time dimension.

Movie Gen Video utilizes the conditional flow matching objective with the same CondOT path (see Algorithm 1). Like Stable Diffusion 3, Movie Gen Video also operates in the latent space of a frozen, pretrained autoencoder. Note that the autoencoder is even more important for videos than for images, which is why most video generators right now are limited in the length of the video they generate.

Specifically, the authors introduce a **temporal autoencoder** (TAE) which maps a raw

[1] In their work, they use a different convention to condition on the noise. But this is only notation and the algorithm is the same.

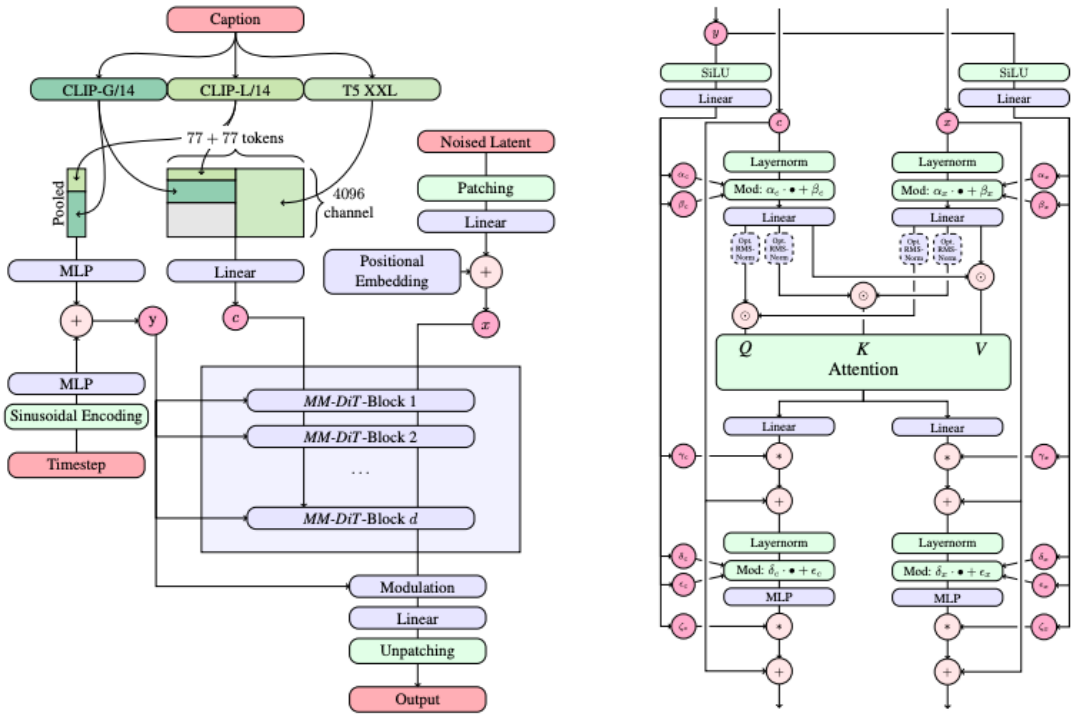

Figure 5: The architecture of the multi-modal diffusion transformer (MM-DiT) proposed in Esser et al. (2024). Figure also taken from Esser et al. (2024).

video $x_t^{(0)} \in \mathbb{R}^{T' \times 3 \times H' \times W'}$ to a latent $x_t \in \mathbb{R}^{T \times C \times H \times W}$, with

$$\frac{T'}{T} = \frac{H'}{H} = \frac{W'}{W} = 8.$$

To accommodate long videos, a temporal tiling procedure is proposed by which the video is chopped up into pieces, each piece is encoded separately, and the latents are stitched together (Polyak et al., 2024).

The model itself (i.e., $u_t^\theta(x_t)$) is given by a DiT-like backbone in which x_t is patchified along the time and space dimensions. The patches are then passed through a transformer employing both self-attention among the patches and cross-attention with language model embeddings, similar to the MM-DiT employed by Stable Diffusion 3.

For text conditioning, Movie Gen Video employs three types of text embeddings: UL2 embeddings (for granular, text-based reasoning) (Tay et al., 2022), ByT5 embeddings (for attending to character-level details, e.g. prompts explicitly requesting specific text to be present) (Xue et al., 2022), and MetaCLIP embeddings trained in a shared text-image embedding space (Lavoie et al., 2024). Their final, largest model has **30 billion parameters**. For a significantly more detailed and expansive treatment, we encourage the reader to check out the Movie Gen technical report itself (Polyak et al., 2024).

References

Dhariwal, P. and Nichol, A. (2021). Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Esser, P., Kulal, S., Blattmann, A., Entezari, R., Müller, J., Saini, H., Levi, Y., Lorenz, D., Sauer, A., Boesel, F., et al. (2024). Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first international conference on machine learning*.

Ho, J. and Salimans, T. (2022). Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*.

Lavoie, S., Kirichenko, P., Ibrahim, M., Assran, M., Wilson, A. G., Courville, A., and Ballas, N. (2024). Modeling caption diversity in contrastive vision-language pretraining. *arXiv preprint arXiv:2405.00740*.

Peebles, W. and Xie, S. (2023). Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4195–4205.

Polyak, A., Zohar, A., Brown, A., Tjandra, A., Sinha, A., Lee, A., Vyas, A., Shi, B., Ma, C.-Y., Chuang, C.-Y., et al. (2024). Movie gen: A cast of media foundation models. *arXiv preprint arXiv:2410.13720*.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. (2021). Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer.

Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E. L., Ghasemipour, K., Gontijo Lopes, R., Karagol Ayan, B., Salimans, T., et al. (2022). Photorealistic text-to-image diffusion models with deep language understanding. *Advances in neural information processing systems*, 35:36479–36494.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. (2020). Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*.

Tay, Y., Dehghani, M., Tran, V. Q., Garcia, X., Wei, J., Wang, X., Chung, H. W., Shakeri, S., Bahri, D., Schuster, T., et al. (2022). Ul2: Unifying language learning paradigms. *arXiv preprint arXiv:2205.05131*.

Vahdat, A., Kreis, K., and Kautz, J. (2021). Score-based generative modeling in latent space. *Advances in neural information processing systems*, 34:11287–11302.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Xue, L., Barua, A., Constant, N., Al-Rfou, R., Narang, S., Kale, M., Roberts, A., and Raffel, C. (2022). Byt5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306.