AI Applications Lecture 11
Introduction to Image Generation AI

SUZUKI, Atsushi
Jing WANG

2025-10-21

Contents

1 Introduction

1.1 Recap of the First Half (Lectures 5–10)

In the first half of this course (Lectures **5** to **10**), we focused on **Natural Language Generation**, covering concepts such as the **pipeline**, **tokenization**, **sampling**, **special tokens**, and **chat templates**. The core perspective was that the **pipeline, as a composition of multiple functions, plays the main role in practical applications, rather than a single neural network**.

1.2 Learning Outcomes of This Lecture

Through this lecture, students will be able to explain the following:

- Explain the mathematical characteristics of the **image generation** task and how it differs from natural language generation.
- Explain what **combination of components** constitutes practical **text-to-image AI**.
- Explain why practical text-to-image AI is structured the way it is, based on its **task characteristics**.

1.3 Revisiting the Grand Principle

The goal of AI is to learn the **appropriate relationship**, i.e., a **function**, between input and output. This is the same for both natural language generation and image generation.

2 Preparation: Mathematical Notations

We reiterate the basic notations used in this lecture.

- **Definition:**
  - $(\mathrm{LHS}) \coloneqq (\mathrm{RHS})$: Indicates that the left-hand side is defined by the right-hand side. For example, $a \coloneqq b$ indicates that a is defined as b.
- **Set:**
  - Sets are often denoted by uppercase calligraphic letters. Example: $\mathcal{A}$.
  - $x \in \mathcal{A}$: Indicates that element x belongs to set $\mathcal{A}$.
  - $\{\}$: The empty set.
  - $\{a, b, c\}$: The set consisting of elements a, b, c (roster notation).

- $\{x \in \mathcal{A} \mid P(x)\}$: The set of elements in $\mathcal{A}$ for which the proposition $P(x)$ is true (set-builder notation).
- $|\mathcal{A}|$: The number of elements in set $\mathcal{A}$ (in this lecture, generally used only for finite sets).
- $\mathbb{R}$: The set of all real numbers.
- $\mathbb{R}_{>0}$: The set of all positive real numbers.
- $\mathbb{R}_{\geq 0}$: The set of all non-negative real numbers.
- $\mathbb{Z}$: The set of all integers.
- $\mathbb{Z}_{>0}$: The set of all positive integers.
- $\mathbb{Z}_{\geq 0}$: The set of all non-negative integers.
- $[1, k]_{\mathbb{Z}}$: When k is a positive integer, $[1, k]_{\mathbb{Z}} \coloneqq \{1, 2, \dots, k\}$, i.e., the set of integers from 1 to k. When $k = +\infty$, $[1, k]_{\mathbb{Z}} \coloneqq \mathbb{Z}_{>0}$, i.e., the set of all positive integers.

• **Function:**

- $f : \mathcal{X} \to \mathcal{Y}$: Indicates that the function f is a map that takes an element from set $\mathcal{X}$ as input and outputs an element from set $\mathcal{Y}$.
- $y = f(x)$: Indicates that the output is $y \in \mathcal{Y}$ when $x \in \mathcal{X}$ is input to function f.

• **Vector:**

- In this course, a vector refers to a column of numbers.
- Vectors are denoted by bold italic lowercase letters. Example: $\boldsymbol{v}$.
- $\boldsymbol{v} \in \mathbb{R}^n$: Indicates that vector $\boldsymbol{v}$ is an n-dimensional real vector.
- The i-th element of vector $\boldsymbol{v}$ is denoted as v_i.

$$\boldsymbol{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}. \tag{1}$$

- For two vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^{d_{\mathrm{emb}}}$, the standard inner product is defined as

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle \coloneqq \sum_{i=1}^{d_{\mathrm{emb}}} u_i v_i. \tag{2}$$

• **Sequence:**

- Given a set $\mathcal{A}$, $n \in \mathbb{Z}_{>0} \cup \{+\infty\}$, and a function $\boldsymbol{a} : [1, n]_{\mathbb{Z}} \to \mathcal{A}$, $\boldsymbol{a}$ is called a sequence of length n consisting of elements from set $\mathcal{A}$. When $n < +\infty$, the sequence is called a finite sequence, and when $n = \infty$, it is called an infinite sequence.

- Sequences are denoted by bold italic lowercase letters, similar to vectors. This is because a finite sequence can be regarded as an extension of a real vector. In fact, a finite sequence of elements from $\mathbb{R}$ can be regarded as a real vector.
- Let $\boldsymbol{a}$ be a sequence of length n with elements from set $\mathcal{A}$. For $i \in [1, n]_{\mathbb{Z}}$, the i-th component a_i is defined as $a_i \coloneqq \boldsymbol{a}(i)$.
- When $n < +\infty$, a sequence $\boldsymbol{a}$ of length n with elements from set $\mathcal{A}$ is determined by its elements $a_1, a_2, ..., a_n$, so we write $\boldsymbol{a} = (a_1, a_2, ..., a_n)$. Similarly, if $\boldsymbol{a}$ is an infinite sequence, we write $\boldsymbol{a} = (a_1, a_2, ...)$.
- The length of a sequence $\boldsymbol{a}$ is written as $|\boldsymbol{a}|$.

- **Matrix:**
  - Matrices are denoted by bold italic uppercase letters. Example: $\boldsymbol{A}$.
  - $\boldsymbol{A} \in \mathbb{R}^{m,n}$: Indicates that matrix $\boldsymbol{A}$ is an $m \times n$ real matrix.
  - The element in the i-th row and j-th column of matrix $\boldsymbol{A}$ is denoted as $a_{i,j}$.

$$\boldsymbol{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}. \tag{3}$$

  - The transpose of matrix $\boldsymbol{A}$ is denoted as $\boldsymbol{A}^\top$. If $\boldsymbol{A} \in \mathbb{R}^{m,n}$, then $\boldsymbol{A}^\top \in \mathbb{R}^{n,m}$, and

$$\boldsymbol{A}^\top = \begin{bmatrix} a_{1,1} & a_{2,1} & \cdots & a_{m,1} \\ a_{1,2} & a_{2,2} & \cdots & a_{m,2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1,n} & a_{2,n} & \cdots & a_{m,n} \end{bmatrix}. \tag{4}$$

  - A vector is also a matrix with 1 column, and its transpose can also be defined.

$$\boldsymbol{v}^\top = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix} \in \mathbb{R}^{1,n}. \tag{5}$$

- **Tensor:**
  - In this lecture, the word tensor simply refers to a multi-dimensional array. A vector can be seen as a first-order tensor, and a matrix as a second-order tensor. Tensors of third order or higher are denoted by underlined bold italic uppercase letters, like $\underline{\boldsymbol{A}}$.

3 Characteristics of Image Generation as a Task

We organize the mathematical characteristics of image generation by comparing it with natural language generation (auto-regressive models).

3.1 Aspects Where Image Generation is Easier

- **Pre-determination of Output Dimensions**: In image output applications, (H, W, C) is often explicitly given by the user or can be fixed, so it is **often unnecessary to dynamically determine** the output length based on the input.
- **Tolerance to Local Errors**: **Local errors** pixel by pixel (e.g., a difference of one intensity level) are often acceptable. There are usually no strong **discrete constraints** on the adjacency of pixels. In contrast, in natural language, strong **local constraints** exist in token sequences due to grammar, etc. (e.g., although ”a” and ”the” are individually frequent, ”the” almost never appears immediately after ”a”).

3.2 Aspects Where Image Generation is Harder

- **Enormous Output Dimensionality**: Even on a small scale, if $H = W = 256,\ C = 3$, the dimensionality is $196{,}608$. In contrast, many natural language applications can be covered with an output of $10{,}000$ tokens.
- **Need for Information Supplementation due to Under-determination**: Particularly in text-to-image, the input (text) is at most a few dozen to a hundred tokens, which is often much less information than the output. For example, suppose the input prompt is the single word ”dog”. For image generation, unless information not in the input prompt —such as the dog’s color, breed, size, and background—is supplemented, the output image cannot be determined. More abstractly, even if the amount of input information is small, it is necessary to generate a large amount of information corresponding to the high dimensionality of the output space, so when supplementing that missing information, there is an arbitrariness to the supplemented information. If this were, for example, a translation task in natural language generation, it would suffice to output the word corresponding to ”dog”, and there would be no need to add other information. Therefore, in the case of text-to-image, **diverse solutions** for the same condition are natural, and **diversity control** by $\mathcal{U}$ becomes essential.

4 Why Probability Mass Function (PMF)-based Methods are Disadvantageous for Image Generation

4.1 Review of NLG Formulation by Conditional PMF

In the standard formulation of natural language generation, a probabilistic language model (conditional probability mass function) over the vocabulary $\mathcal{V}$

$$P(y_t \mid \boldsymbol{y}_{<t}, \boldsymbol{x}), \quad t = 1, 2, \ldots \tag{6}$$

is represented by a neural network, and a sequence is generated by **sequential sampling**. Output can be stopped by the appearance of **EOS (end-of-sequence)**, allowing the output length to be determined **dynamically**. Furthermore, **elimination of inconsistent tokens** (utilization of local constraints) is possible.

4.2 Difficulty of Application to Image Generation

Applying similar sequential sampling to image generation requires sampling HWC dimensions **one component at a time**, leading to an enormous computational cost. Also, since images have **weak local constraints**, the advantages of Eq. (6) are not well leveraged.

Furthermore, attempting to represent the joint distribution directly with a neural network, rather than a conditional distribution, is impractical as the number of elements in the product space is astronomical.

5 Practical Pipeline for text-to-image

As we have seen so far, in text-to-image, the input prompt rarely contains enough information to sufficiently narrow down the output candidates, so we want to probabilistically assign diverse outputs to a single input. However, it is impractical to assign probability mass to all possible output values, as is done in natural language output. On the other hand, images, as data, are not discrete, and the real-valued output returned by the neural network is acceptable as is, so there is no necessity to perform sampling on the output side. Therefore, in practical text-to-image systems, random seed dependency is placed on the input side from the neural network's perspective. This is in contrast to natural language generation, which places random seed dependency on the output side from the neural network's perspective. Below, we formulate text-to-image in a way that depends on a random seed and explain the configuration of a practical pipeline to achieve this.

5.1 Problem Formulation

In the following, a **string** is assumed to be an element of $\mathcal{V}^*$, the set of finite sequences over the token set $\mathcal{V}$. Let the input be one or more natural language strings $(\boldsymbol{c}^{(i)})_{i=1}^{m} \in (\mathcal{V}^*)^m$ (where m is the number of strings), and the output be an image $\underline{\boldsymbol{X}}$ (e.g., an $H \times W \times C$ real-valued tensor). **text-to-image** is a task that aims to generate one of the images matching the information contained in the string(s), given one or more strings and a random seed as input, and is equivalent to learning some parametric function

$$\mathsf{Text2Img}_{\boldsymbol{\theta}} : (\mathcal{V}^*)^m \times \mathcal{U} \to \mathcal{I} \tag{7}$$

Here, $\mathcal{U}$ is the space of random seeds (initial values for pseudo-random numbers) or noise sequences, and $\mathcal{I} \subset \mathbb{R}^{H\times W\times C}$ is the image space.

The output image $\underline{\boldsymbol{X}}$ is represented by

$$\underline{\boldsymbol{X}} = \mathsf{Text2Img}_{\boldsymbol{\theta}}\left((\boldsymbol{c}^{(i)})_{i=1}^{m}, \boldsymbol{u}\right). \tag{8}$$

The ideal map $\mathsf{Text2Img}_{\boldsymbol{\theta}}$ is one that outputs different images corresponding to the information in the input strings by changing the random seed.

Remark 5.1. The goal of Eq. (7) is to obtain **diverse realizations** by scanning $\mathcal{U}$, while having a **randomness-invariant core structure** (a semantically consistent group of images for the same condition).

5.2 General-Purpose Configuration in a Practical Pipeline

It is sufficient to construct a function that fits Eq. (7) and Eq. (8), without **directly** obtaining the probability mass function of the output space. In practical systems (e.g., **Latent Diffusion Models** [4]), the following **three-layer configuration** is widely used.

1. **Text encoder** $\mathsf{TextEnc}^{(i)}_{\boldsymbol{\alpha}^{(i)}} : \mathcal{V}^* \to \mathbb{R}^{d^{(i)}}$. For multiple string inputs $(\boldsymbol{c}^{(i)})_{i=1}^{m}$, it outputs embedding sequences $(\boldsymbol{e}^{(i)})_{i=1}^{m}$. Here, each $\boldsymbol{e}^{(i)} \in \mathbb{R}^{d^{(i)}}$, and $d^{(i)}$ may vary depending on the input. Example: **CLIP** text encoder [3].

2. **Low-resolution "image" generator (generator in latent space)** $\mathsf{LatentGen}_{\boldsymbol{\beta}} : \left(\prod_{i=1}^{m} \mathbb{R}^{d^{(i)}}\right) \times \mathcal{U} \to \mathcal{Z}$. Here $\mathcal{Z} \subset \mathbb{R}^{h\times w\times c}$ is the **latent space**. A typical example is implementing the **reverse diffusion process** [1] of **diffusion models** using a **U-Net** [5].

3. **Decoder** $\mathsf{Dec}_{\gamma} : \mathcal{Z} \to \mathcal{I}$. Example: The decoder of a **Variational Autoencoder (VAE)** [2].

As a whole,

$$\mathsf{Text2Img}_{\boldsymbol{\theta}}\left((\boldsymbol{c}^{(i)})_{i=1}^{m}, \boldsymbol{u}\right) := \mathsf{Dec}_{\gamma}\left(\mathsf{LatentGen}_{\boldsymbol{\beta}}\left((\mathsf{TextEnc}^{(i)}_{\boldsymbol{\alpha}^{(i)}}(\boldsymbol{c}^{(i)}))_{i=1}^{m}, \boldsymbol{u}\right)\right) \tag{9}$$

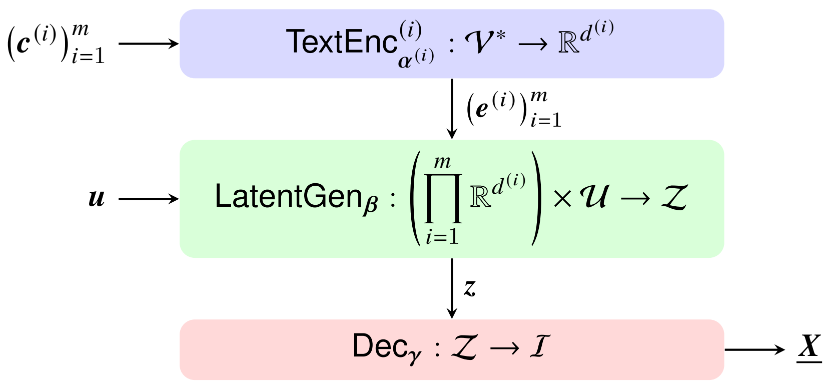
$(c^{(i)})_{i=1}^m$
$\mathrm{TextEnc}_{\alpha^{(i)}}^{(i)} : \mathcal{V}^* \to \mathbb{R}^{d^{(i)}}$
$(e^{(i)})_{i=1}^m$
u
$\mathrm{LatentGen}_\beta : \left(\prod_{i=1}^m \mathbb{R}^{d^{(i)}}\right) \times \mathcal{U} \to \mathcal{Z}$
z
$\mathrm{Dec}_\gamma : \mathcal{Z} \to \mathcal{I}$
$\underline{X}$

Figure 1: Three-layer configuration of the text-to-image pipeline (top-to-bottom flow, borderless light-colored blocks).

When there are multiple condition inputs $\left(c^{(i)}\right)_{i=1}^m$,

$$\mathrm{Text2Img}_\theta\left(\left(c^{(i)}\right)_{i=1}^m, u\right) := \mathrm{Dec}_\gamma\left(\mathrm{LatentGen}_\beta\left(\left(e^{(i)}\right)_{i=1}^m, u\right)\right), \quad e^{(i)} := \mathrm{TextEnc}_{\alpha^{(i)}}^{(i)}\left(c^{(i)}\right). \tag{10}$$

Remark 5.2. TextEnc and Dec can be **reused** from existing models, making it easy to **focus** learning on $\mathrm{LatentGen}_\theta$. This improves data efficiency and development efficiency.

5.3 Overview of Stable Diffusion 1.5

As a representative example, we overview **Stable Diffusion v1.5** (academically, one implementation of **Latent Diffusion Models** [4]).

- TextEnc: **CLIP** text encoder (Transformer-based) [3]. Conditioning is provided to the U-Net [5] via **cross-attention** [4].
- $\mathrm{LatentGen}_\theta$: Generates the latent $\mathcal{Z}$ by **reverse diffusion**, iteratively applying **noise prediction** by a **U-Net** [5] [1,4].
- Dec: Reconstructs from latent to pixel space with a **VAE decoder** [2,4].

5.4 Alignment of this Configuration with Task Characteristics

- **Batch Update of Enormous Output**: Computation is efficient because it **updates high dimensions all at once** without sequential tokenization (applying U-Net once per step).
- **Handling Under-determination**: **Diversity control** via $\mathcal{U}$ is built-in, and diverse solutions from the same condition can be generated naturally.
- **Reduction of Computational Cost**: Instead of outputting a high-dimensional image, the process outputs a low-dimensional latent variable and then converts it to a high-dimensional image with Dec, thereby **reducing computational cost**.

Remark 5.3 (Difference between Research Paper and General Implementation in Stable Diffusion 1.5)**.** Note the **difference between the research paper and general implementations**. The original paper on **Latent Diffusion Models** [4] explicitly states that the domain-specific encoder τ_θ (e.g., text encoder) for conditioning is **trained jointly** with the reverse diffusion network ϵ_θ. In fact, the paper states, **”both τ_θ and ϵ_θ are jointly optimized via Eq. 3.”** and **”implement τ_θ as a transformer ... to infer a latent code”** (Sec. 3.3 and Sec. 4.3.1). On the other hand, the **Stable Diffusion** implementation in **Hugging Face Diffusers**, which is widely used in practice, states, **”This model uses a frozen CLIP ViT-L/14 text encoder to condition the model on text prompts.”**[a], and uses the text encoder as **frozen**. That is, there is an **operational difference** between the design principle of the original paper (whether it is jointly trained) and the representative implementation (using a fixed, existing encoder).

[a]`https://huggingface.co/docs/diffusers/api/pipelines/stable_diffusion/overview`

5.5 Maintaining Consistency of Multiple Components in Latent Space

The latent space $\mathcal{Z}$ is the output space for the low-resolution ’image’ generator LatentGen$_\beta$, which receives information from the string input, and the input space for the decoder Dec$_\gamma$. The latent space connects these two components. Since neither component can be expected to learn a perfectly ideal function, each component should have the following properties with respect to the latent space so that it can accommodate the other’s deviations.

- As the output space of the text encoder: When two sentences with similar content to be reflected in the image are input, their respective outputs are close to each other.
- As the input space of the decoder: When two inputs that are close to each other are given, their respective outputs are close to each other.

Satisfying these allows the low-resolution ’image’ generator to obtain an output close to the ideal, even if its output is not perfectly ideal. This leads to ease of learning and stability in inference.

6 Summary and Future Plans

6.1 Summary Corresponding to Learning Outcomes

- **Task Characteristics**: Image generation has **enormous output dimensions**, **weak local constraints**, and is **under-determined**.
- **Pipeline Configuration**: Practical text-to-image consists of a **text encoder** TextEnc, a **latent generator** LatentGen$_\theta$ (reverse diffusion + U-Net), and a **decoder** Dec (VAE).

- **Reason for Configuration**: It **aligns with the task characteristics** because it updates high dimensions collectively and continuously, controlling diversity with randomness while maintaining semantic consistency.

6.2 Next Lecture Preview

In the next and subsequent lectures, we will detail each component. Naturally, each component must be consistent with the others. First, we will explain the **VAE** (**encoder/decoder**, latent regularization, inference, and learning), which connects the image space and the latent space at the outermost layer.

References

[1] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In Advances in Neural Information Processing Systems (NeurIPS), 2020.

[2] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114, 2013.

[3] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamila Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In Proceedings of the 38th International Conference on Machine Learning (ICML), 2021.

[4] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 10684–10695, 2022.

[5] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Medical Image Computing and Computer-Assisted Intervention (MICCAI), pages 234–241, 2015.