# HOMEWORK 4
# MULTI-MODAL FOUNDATION MODELS *

10-423/10-623 GENERATIVE AI
http://423.mlcourse.org

OUT: Oct. 25, 2024
DUE: Nov. 5, 2024
TAs: Ryan, Haoyang, Shrikara

## Instructions

- **Collaboration Policy**: Please read the collaboration policy in the syllabus.
- **Late Submission Policy:** See the late submission policy in the syllabus.
- **Submitting your work:** You will use Gradescope to submit answers to all questions and code.
  - **Written:** You will submit your completed homework as a PDF to Gradescope. Please use the provided template. Submissions can be handwritten, but must be clearly legible; otherwise, you will not be awarded marks. Alternatively, submissions can be written in LaTeX. Each answer should be within the box provided. If you do not follow the template, your assignment may not be graded correctly by our AI assisted grader and there will be a **2% penalty** (e.g., if the homework is out of 100 points, 2 points will be deducted from your final score).
  - **Programming:** You will submit your code for programming questions to Gradescope. We will examine your code by hand and may award marks for its submission.
- **Materials:** The data that you will need in order to complete this assignment is posted along with the writeup and template on the course website.

| Question | Points |
|---|---|
| LaTeX Template Alignment | 0 |
| Latent Diffusion Model (LDM) | 6 |
| VQ-VAEs | 6 |
| CLIP | 4 |
| Programming: Prompt2Prompt | 38 |
| Code Upload | 0 |
| Collaboration Questions | 2 |
| Total: | 56 |

*Compiled on Friday 25th October, 2024 at 12:00

# 1 LaTeX Template Alignment (0 points)

1.1. (0 points) **Select one:** Did you use LaTeX for the entire written portion of this homework?

◯ Yes

◯ No

1.2. (0 points) **Select one:** I have ensured that my final submission is aligned with the original template given to me in the handout file and that I haven't deleted or resized any items or made any other modifications which will result in a misaligned template. I understand that incorrectly responding yes to this question will result in a penalty equivalent to 2% of the points on this assignment.
**Note:** Failing to answer this question will not exempt you from the 2% misalignment penalty.

◯ Yes

# 2 Latent Diffusion Model (LDM) (6 points)

2.1. (2 points) **Short answer:** Why does a latent diffusion model run diffusion in a latent space instead of pixel space?

2.2. **Short answer:** Standard cross-attention for a diffusion-based text-to-image model defines the queries $\mathbf{Q}$ as a function of the pixels (or latent space) $\mathbf{Y} \in \mathbb{R}^{m \times d_y}$, and the keys $\mathbf{K}$ and values $\mathbf{V}$ as a function of the text encoder output $\mathbf{X} \in \mathbb{R}^{n \times d_x}$.

$$\mathbf{Q} = \mathbf{Y}\mathbf{W}_q, \qquad \mathbf{K} = \mathbf{X}\mathbf{W}_k, \qquad \mathbf{V} = \mathbf{X}\mathbf{W}_v$$

(where $\mathbf{W}_q \in \mathbb{R}^{d_y \times d}$ and $\mathbf{W}_k, \mathbf{W}_v \in \mathbb{R}^{d_x \times d}$) and then applies standard attention:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{Q}\mathbf{K}^T/\sqrt{d})\mathbf{V}$$

Now, suppose you instead defined a new formulation where the values are a function of the pixels (or latent space): $\mathbf{V} = \mathbf{Y}\mathbf{W}_v$ where $\mathbf{W}_v \in \mathbb{R}^{d_y \times d}$.

2.2.a. (2 points) What goes wrong mathematically in the new formulation?

2.2.b. (2 points) Intuitively, why doesn't the new formulation make sense? Briefly begin with an explanation of what the original formulation of cross-attention is trying to accomplish for a single query vector, and why this new formulation fails to accomplish that.

# 3 VQ-VAEs (6 points)

3.1. The objective function for a VQ-VAE contains two terms in addition to the reconstruction loss term:

- The vector-quanization loss, $\|\text{sg}\left[z_e(x)\right] - e\|_2^2$ and
- The "commitment" loss, $\beta\|z_e(x) - \text{sg}\left[e\right]\|_2^2$

where sg is the stopgradient operator and $e$ is the latent embedding vector closest to the output of the encoder $z_e(x)$.[1]

In this question, you will examine the impact of these terms and the stopgradient operator. Let

$$\mathcal{L} = \|\text{sg}\left[z_e(x)\right] - e\|_2^2 + \beta\|z_e(x) - \text{sg}\left[e\right]\|_2^2 \quad (1)$$
$$\tilde{\mathcal{L}} = \|z_e(x) - e\|_2^2 + \beta\|z_e(x) - e\|_2^2 \quad (2)$$

3.1.a. (1 point) **Math:** What is the gradient of $\mathcal{L}$ with respect to $z_e(x)$?

3.1.b. (1 point) **Math:** What is the gradient of $\mathcal{L}$ with respect to $e$?

3.1.c. (1 point) **Math:** What is the gradient of $\tilde{\mathcal{L}}$ with respect to $z_e(x)$?

3.1.d. (1 point) **Math:** What is the gradient of $\tilde{\mathcal{L}}$ with respect to $e$?

3.1.e. (2 points) **Short answer:** Given your findings from the previous parts, how would you describe the impact of the stopgradient operator on the optimization of these two terms? How do the gradients with and without the stopgradient operator differ?

[1] For the purposes of this question, we are still assuming that the encoder outputs a single vector; in practice, a true VQ-VAE encoder would output multiple vectors and these terms in the objective function would be sums over these vectors.

# 4 CLIP (4 points)

4.1. (4 points) **Short answer:** In https://arxiv.org/pdf/2103.00020, Radford et al., pre-trained the image and text encoders using a *symmetric cross-entropy loss*. Formally, given a mini-batch of $N$ (image, caption) pairs, let $e_i^{(n)}$ be the image embedding of the $n^{\text{th}}$ image normalized to have unit norm and let $e_t^{(n)}$ be the text embedding of the $n^{\text{th}}$ caption, also normalized to have unit norm.

The symmetric cross-entropy loss consists of two terms for each (image, caption) pair, a term that compares the image embedding to all $N$ caption embeddings and a term that compares the caption embedding to all $N$ image embeddings. Formally, let

$$\ell_{i\to t}^{(n)} = -\log\left(\frac{\exp\left(e_i^{(n)} \cdot e_t^{(n)}\right)}{\sum_{m=1}^{N}\exp\left(e_i^{(n)} \cdot e_t^{(m)}\right)}\right) \tag{3}$$

$$\ell_{t\to i}^{(n)} = -\log\left(\frac{\exp\left(e_i^{(n)} \cdot e_t^{(n)}\right)}{\sum_{m=1}^{N}\exp\left(e_i^{(m)} \cdot e_t^{(n)}\right)}\right) \tag{4}$$

where $\cdot$ is the dot-product operation between two (same-length) vectors. The CLIP objective can then be expressed as

$$\ell = \frac{1}{N}\sum_{n=1}^{N}\frac{\ell_{i\to t}^{(n)} + \ell_{t\to i}^{(n)}}{2} \tag{5}$$

Show that minimizing this objective function is equivalent to jointly maximizing the cosine similarity of an image embedding and its corresponding caption embedding while minimizing the cosine similarity of all other pairs of image and caption embeddings. You may use (clear, concise) English sentences or mathematical formulae or both in your response.

# 5 Programming: Prompt2Prompt (38 points)

Reminder: you are not permitted to use code from any existing implementations of prompt-to-prompt.

## Introduction

In this section, we explore an innovative approach to image editing. Editing techniques aim to retain the majority of the original image's content while making certain changes. However, current text-to-image models often produce completely different images when only a minor change to the prompt is made. State-of-the-art methods typically require a spatial mask to indicate the modification area, which ignores the original image's structure and content in that region, resulting in significant information loss.

In contrast, the Prompt2Prompt framework by Hertz et al. (2022) facilitates edits using only text, striving to preserve original image elements while allowing for changes in specific areas.

Cross-attention maps, which are high-dimensional tensors binding pixels with prompt text tokens, hold rich semantic relationships crucial to image generation. The key idea is to edit the image by injecting these maps into the diffusion process. This method controls which pixels relate to which particular prompt text tokens throughout the diffusion steps, allowing for targeted image modifications.

You'll explore modifying token values to change scene elements (e.g. a "dog" riding a bicycle → a "cat" riding a bicycle) while maintaining the original cross-attention maps to keep the scene's layout intact.

## HuggingFace Diffusers

In this assignment, we will be using HuggingFace's diffusers, a library created for easily using well-known state-of-the-art diffusion models, including creating the model classes, loading pre-trained weights, and calling specific parts of the models for inference. Specifically, we will be using the API for the class `DiffusionPipeline` and methods from its subclass `LDMTextToImagePipeline` for loading the pre-trained LDM model.

You are required to read the API for LDMTextToImagePipeline:

https://huggingface.co/docs/diffusers/en/api/pipelines/latent_diffusion

You will be implementing the model loading and calling individual components of LDMTextToImagePipeline in this assignment.

## Starter Code

The files are organized as follows:

```
hw4/
  run_in_colab.ipynb
  prompt2prompt.py
  requirements.txt
```

Here is what you will find in each file:

1. `run_in_colab.ipynb`: This is where you can run inference and see the visualization of your implemented methods.

2. `prompt2prompt.py`: Contains the follwoing classes and helper functions:
   (a) Class `MyLDMPipeline` - Initializes the pipeline by downloading the pre-trained model. Contains the method `_generate_image_from_text` which uses the swapper from `MySharedAttentionSwapper` to change the attention and generate new images from the prompt. This class also contains the method `get_random_noise`, which needs to be implemented. This is used to generate the latents required for image generation
   (b) Method `get_replacement_mapper_(...)` - Details of the function provided below. Locations in the code where changes ought to be made are marked with a TODO.
   (c) Class `MySharedAttentionSwapper` - contains the method `swap_attention_probs(...)` which needs to be implemented. This method is responsible for swapping attention probabilities based on the current state of the model.
   (d) Class `MyCrossAttention` - the actual class that you are going to be using to replace the existing attention of the UNet module. You need to implement the actual attention mechanism (define query, key and value, apply linear projection, dropout and add residual connection, if needed, all in order to calculate the hidden states)

   Carefully read through the entire file to understand the helper functions and their functionalities.
3. `requirements.txt`: A list of packages that need to be installed for this homework.

Carefully read through the entire file to understand the helper functions and their functionalities.

## Command Line

We recommend conducting your final experiments for this homework on Colab. Colab provides a free T4 GPU for code execution.

```
(Run the run_in_colab.ipynb for visualization.)
```

## Prompt2Prompt

In this problem, you will implement Prompt2Prompt in the file `prompt2prompt.py`.

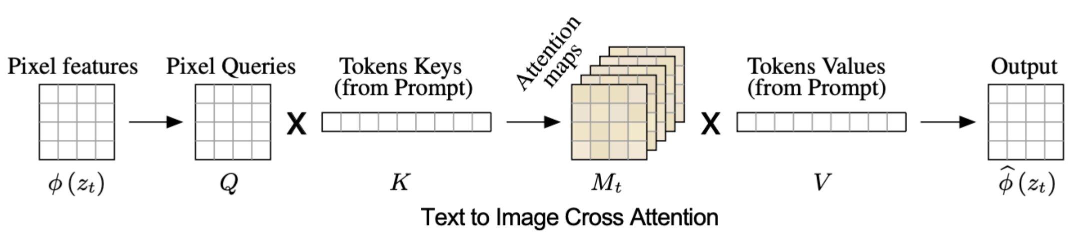


Figure 1: Visual and textual embedding are fused using cross-attention layers that produce attention maps for each textual token. Figure source: Hertz et al. (2022)

### Latent Diffusion Model Pipeline:

This is the class that implements and runs the entire pipeline of prompt2prompt. The `__init__` function loads the different parts of the pre-trained model. The function `get_image_from_text` is the entry point to this class which, as the name suggests, returns the generated image. It internally calls `get_image_from_text_`.

Here is an overview of the key steps this method performs:

- Tokenization and Embedding of Prompts: The model's tokenizer converts both an empty string (to represent the unconditional generation case) and the actual text prompts into tokenized inputs. These tokenized inputs are then passed through a BERT-like model to obtain embeddings. The embeddings for the unconditional inputs and the text prompts are concatenated to serve as the context for the diffusion process.
- Latent Space Initialization: You need to implement `get_random_noise` that achieves this. It needs to initialize a latent space with the specified dimensions. This space will evolve into the final image through the diffusion process.
- Diffusion Process: The core of the image generation happens here. For each timestep defined by num_inference_steps, the function performs a diffusion step. This involves manipulating the latent space towards the desired outcome based on the context and the current timestep, under the guidance of the specified scale. The controller plays a role here in directing the attention mechanism during these steps.
- Image Generation: After completing the diffusion steps, the final latent representation is converted into an image using the model's VQ-VAE (Vector Quantized Variational AutoEncoder).

**Cross Attention:**

The LDM utilizes text prompts to influence the noise prediction at each diffusion step through cross-attention layers. Essentially, at each step $t$, the model predicts noise $\epsilon$ based on a noisy image $z_t$ and the text prompt's embedding $\psi(P)$ using a U-net architecture, leading to the final image $I = z_0$. The key interaction between image and text occurs in the noise prediction phase, where visual and textual embeddings are integrated via cross-attention layers. As illustrated in Fig. 1, these layers generate spatial attention maps for textual tokens by projecting the image's deep features and text embedding into query ($Q$), key ($K$), and value ($V$) matrices through learned projections $\ell_Q, \ell_K, \ell_V$. The attention mechanism is formulated as:

$$M = \text{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right), \tag{6}$$

where $M_{ij}$ represents the influence of the $j$-th token's value on the $i$-th pixel, with $d_k$ being the dimensionality of the keys and queries. The output from cross-attention, $\phi_b(z_t) = MV$, updates the image features $\phi(z_t)$. Intuitively, $MV$ is a weighted average of $V$ based on the attention maps $M$, which are correlated to the similarity between Q and K. This process leverages multi-head attention to enhance expressiveness, concatenating the outcomes from parallel heads and refining them through an additional linear layer for the final output.

**Controlling Cross Attention:**

Pixels are more attracted (correlated) to the words that describe them (you will visualize this when you run the notebook). Building on the insight that cross-attention maps dictate the spatial layout and relationship between pixels and their corresponding descriptive words, Prompt2Prompt proposes a method to edit images while maintaining their original structure. By reusing attention maps $M$ from an initial generation with prompt $P$ in a subsequent generation with an altered prompt $P^*$, we can create an edited image $I^*$ that respects the original image's layout $I$.

We can define $DM(z_t, P, t, s)$ as the function for a single diffusion step $t$, outputting a noisy image $z_{t-1}$ and optionally an attention map $M_t$. We denote $DM(z_t, P, t, s)\{M \leftarrow \hat{M}\}$ to indicate the diffusion step with an externally supplied attention map $\hat{M}$ overriding the attention map $M$, while maintaining the value matrix $V$ from $P$. The attention map generated with the edited prompt $P^*$ is $M_t^*$. The function Edit$(M_t, M_t^*, t)$ represents an editing operation on the attention maps of the original and edited prompts at step $t$. This general algo is written out in Fig. 2.

**Algorithm 1:** Prompt-to-Prompt image editing

**Input:** A source prompt $\mathcal{P}$, a target prompt $\mathcal{P}^*$, and a random seed $s$.<br>
**Optional for local editing:** $w$ and $w^*$, words in $\mathcal{P}$ and $\mathcal{P}^*$, specifying the editing region.<br>
**Output:** A source image $x_{src}$ and an edited image $x_{dst}$.<br>
$z_T \sim N(0, I)$ a unit Gaussian random variable with random seed $s$;<br>
$z_T^* \leftarrow z_T$;<br>
**for** $t = T, T-1, \ldots, 1$ **do**<br>
    $z_{t-1}, M_t \leftarrow DM(z_t, \mathcal{P}, t, s)$;<br>
    $M_t^* \leftarrow DM(z_t^*, \mathcal{P}^*, t, s)$;<br>
    $\widehat{M}_t \leftarrow Edit(M_t, M_t^*, t)$;<br>
    $z_{t-1}^* \leftarrow DM(z_t^*, \mathcal{P}^*, t, s)\{M \leftarrow \widehat{M}_t\}$;<br>
    **if** *local* **then**<br>
        $\alpha \leftarrow B(\overline{M}_{t,w}) \cup B(\overline{M}^*_{t,w^*})$;<br>
        $z_{t-1}^* \leftarrow (1-\alpha) \odot z_{t-1} + \alpha \odot z_{t-1}^*$;<br>
    **end**<br>
**end**<br>
**Return** $(z_0, z_0^*)$

Figure 2: Algorithm: Prompt-to-Prompt image editing. Source: Hertz et al. (2022). Note that *local* is always False in our implementation.

**Word Swap:**

While Prompt-to-Prompt can be used for various different types of edit operations on the prompt, we will focus exclusively on word swapping, e.g., $P$ = “a big bicycle” to $P^*$= “a big car”.

For word swapping, we inject the attention maps of the source image into the generation by the modified prompt. We work with the `MySharedAttentionSwapper` class, where you will initialize a mapper tensor as `self.mapper`, and with the `MyCrossAttentionClass`. It is designed to facilitate the replacement of tokens in the cross-attention map and should be used to reassign attention from the old tokens to the new ones (dive into the code base to see what exactly it does and also refer to the section on Replacement Mapper). You will implement:

- one line of code in `swap_attention_probs(...)` in the `MySharedAttentionSwapper` class where you swap out the existing attention with the `self.mapper`
- `forward` method of the `MyCrossAttentionClass`: In this method you need to initialize and implement the attention mechanism, which means you need to initialize your query, key and value tensors, and use them to calculate the attention probabilities. Once the prompt-to-prompt attention swapping is done, you need to calculate the hidden states and apply the linear projection, dropout and if required, add the residual connection.
  Some functions that might come in handy are:
  - `self.attn.to_q`, `self.attn.to_k` and `self.attn.to_v`: these are linear trans-

formations that project the input into query, key, and value vectors, which are used to compute attention scores and weighted outputs in the attention mechanism.

- `torch.bmm`: performs a batch-wise matrix multiplication of two 3D tensors. Specifically, if you have tensors of shape (b, n, m) and (b, m, p), `torch.bmm` multiplies each pair of matrices in the batch (of size b) to produce a new tensor of shape (b, n, p).

**Replacement Mapper:**

In the function `get_replacement_mapper`, we return the stacked PyTorch tensor containing all the mapping matrices, where each matrix corresponds to the mapping from the first prompt to one of the subsequent prompts. It calls upon `get_replacement_mapper_` (which you will implement) that splits both input strings `x` and `y` into words and constructs a mapping matrix of size `max_len` $\times$ `max_len`, with values in $[0, 1]$ indicating the matching between the changing word in the input prompt and the corresponding word in the modification prompt.

(Hint: For most things in PyTorch we avoid for loops, but you needn't do so here. Since this method is only called once during initialization, for loops are fine.)

**Evaluation:**

We ask you to run the notebook to get the visualizations once you complete filling in the needed functions. You will be visualizing replacement edit and local editing results.

## Empirical Questions

5.1. (1 point) **Short answer:** Print out the structure of the UNet model by calling `print(pipe.unet)`. Notice that within the *up* and *down* transformer blocks (`BasicTransformerBlock`), there are two attention layers: one is named `attn1` and the other `attn2`. Which of these two is self-attention and which is cross-attention? **Briefly justify your answer based on the printout of the model.**

The questions below refer directly to the section headers of the Colab notebook in `run_in_colab.ipynb`.

5.2. (4 points) Paste the results from the section 'Baseline: Different Initial Noise for Each Prompt'

[Expected runtime on Colab T4: 30s]

5.3. (4 points) Paste the results from the section 'Same Initial Noise for Each Prompt'

[Expected runtime on Colab T4: 30s]

5.4. (1 point) Briefly explain how your results from Question 5.2 differ from your results in Question 5.3?

5.5. (4 points) Paste the results from the section 'Prompt-to-Prompt: Word Swap'

[Expected runtime on Colab T4: 30s]

5.6. (1 point) Briefly explain how your results from Question 5.3 differ from your results in Question 5.5?

5.7. (4 points) Paste the results from the section 'Prompt-to-Prompt: Modify Cross-Attention Injection'

[Expected runtime on Colab T4: 30s]

5.8. (2 points) How do you your results in Question 5.7 vary as you change the cross attention parameters?

5.9. (4 points) Paste the results from the section 'Prompt-to-Prompt: Modify Self-Attention Injection'

[Expected runtime on Colab T4: 30s]

5.10. (2 points) How do you your results in Question 5.9 vary as you change the cross attention parameters?

5.11. (4 points) Paste the results from the section 'Prompt-to-Prompt: Single-Token to Multiple-Token Word Swap'

[Expected runtime on Colab T4: 30s]

5.12. (4 points) Paste the results from the section 'Prompt-to-Prompt: Multiple-Token to Single-Token Word Swap'

[Expected runtime on Colab T4: 30s]

5.13. Define your own base prompt and three prompt edits (i.e. something other than the examples provided in the `.ipynb`) and run them through Prompt-to-Prompt.

5.13.a. (1 point) Report the prompts and any hyperparameters that you used.

5.13.b. (2 points) Paste the resulting images below.

# 6 Code Upload (0 points)

6.1. (0 points) Did you upload your code to the appropriate programming slot on Gradescope?
*Hint:* The correct answer is 'yes'.

◯ Yes

◯ No

For this homework, you should upload only `prompt2prompt.py`.

# 7 Collaboration Questions (2 points)

After you have completed all other components of this assignment, report your answers to these questions regarding the collaboration policy. Details of the policy can be found in the syllabus.

7.1. (1 point) Did you collaborate with anyone on this assignment? If so, list their name or Andrew ID and which problems you worked together on.

7.2. (1 point) Did you find or come across code that implements any part of this assignment? If so, include full details.