10-423/10-623 Generative AI

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Prompt to Prompt

Matt Gormley
Lecture 13.5
Mar. 15, 2024

CONDITIONAL IMAGE GENERATION

Image Generation

- Class-conditional generation
- Super resolution
- Image Editing
- Style transfer
- Text-to-image (TTI) generation

Figure from Razavi et al. (2019)

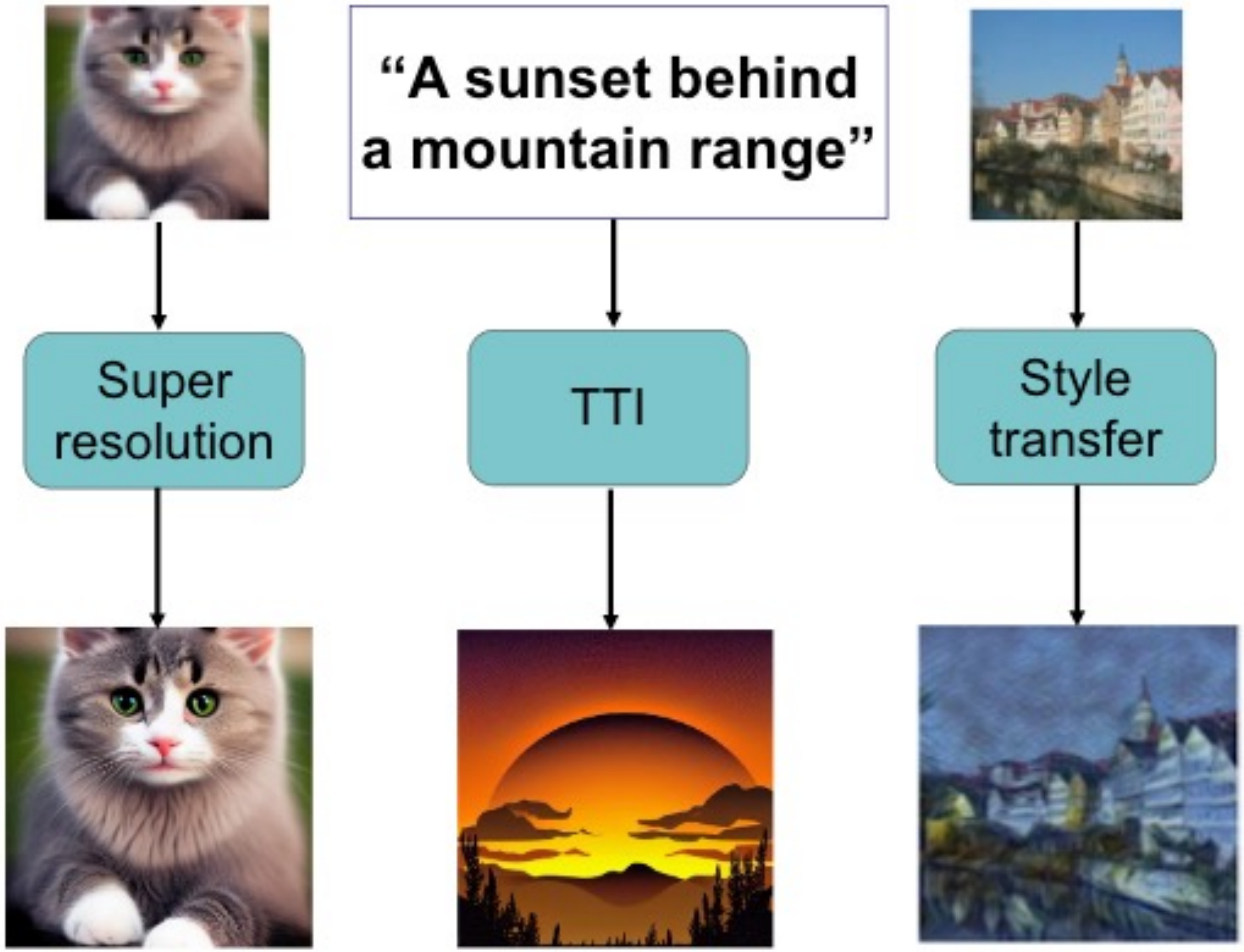


Figure from Bie et al. (2023)

Image Editing

A variety of tasks involve automatic editing of an image:

- **Inpainting** fills in the (pre-specified) missing pixels
- **Colorization** restores color to a greyscale image
- **Uncropping** creates a photo-realistic reconstruction of a missing side of an image

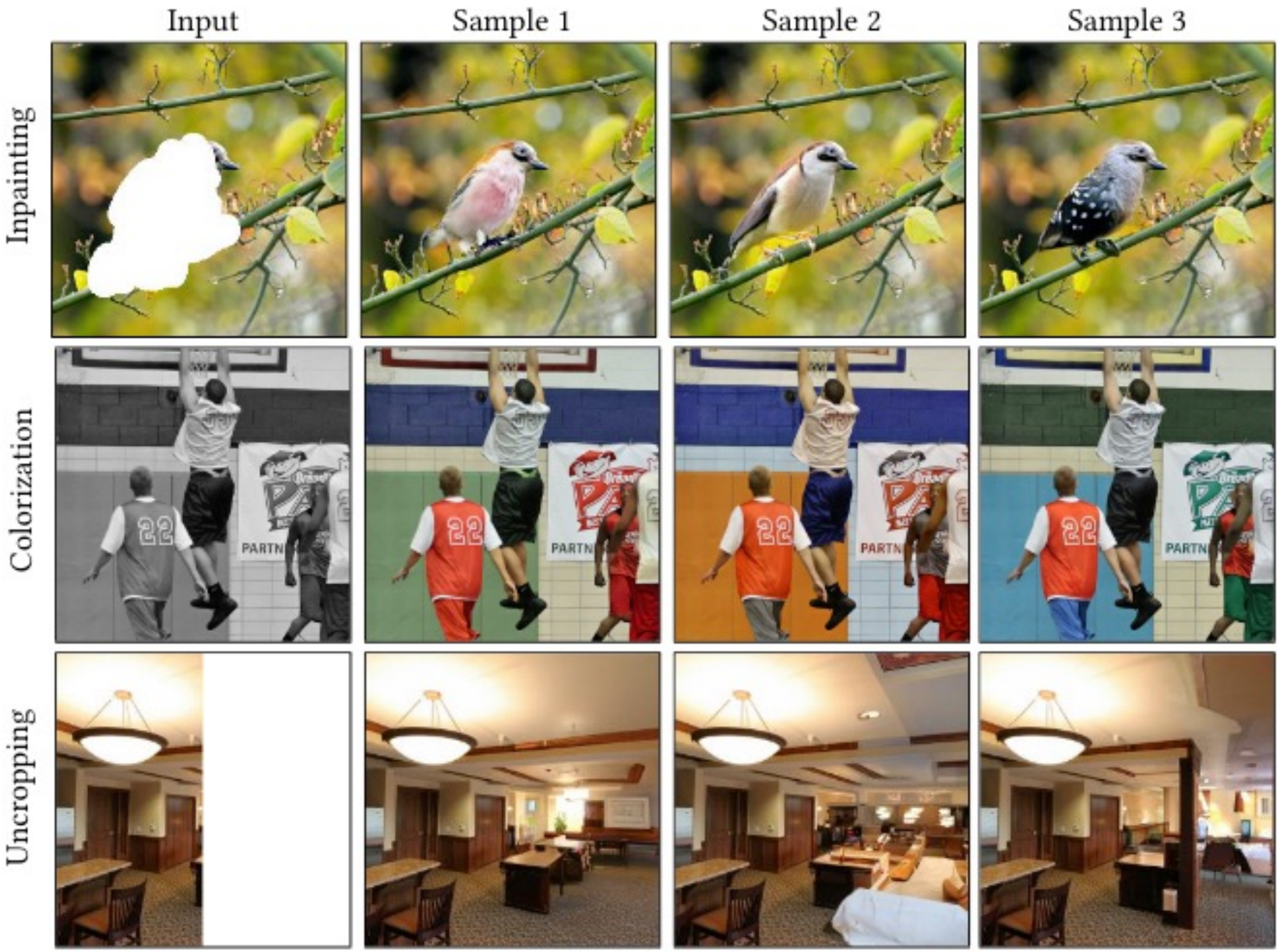
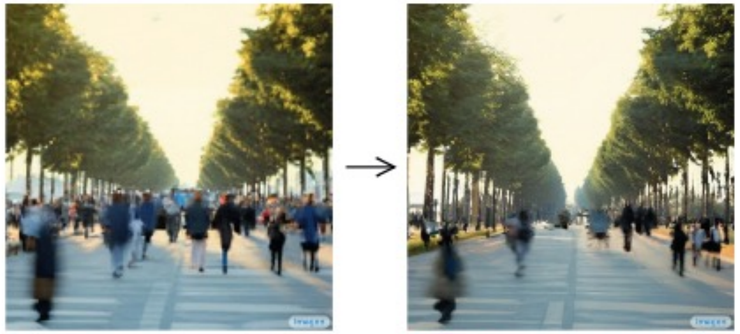


Figure from Saharia et al. (2022)

Editing Images with Text

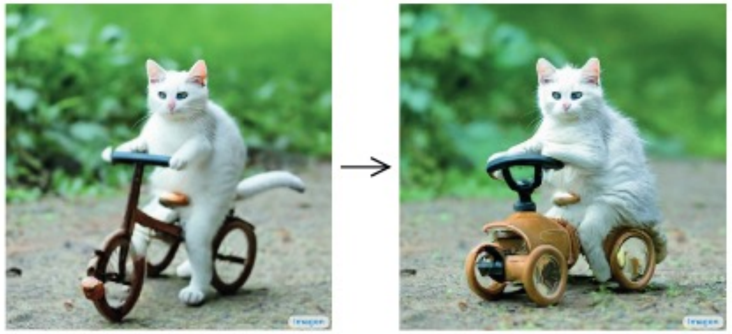
prompt-to-prompt can edit one generated image simply by adjusting the prompt

down-weight existing descriptor in the prompt

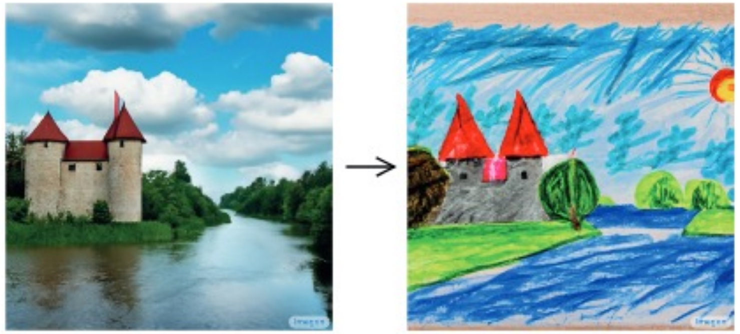


“The boulevards are crowded today.”

swap one word for another



“Photo of a cat riding on a bicycle.”
car



“Children drawing of a castle next to a river.”

phrase insertion for style change

“a cake with decorations.”
jelly beans

phrase insertion for content change

LATENT DIFFUSION MODEL (LDM)

Latent Diffusion Model

Motivation:

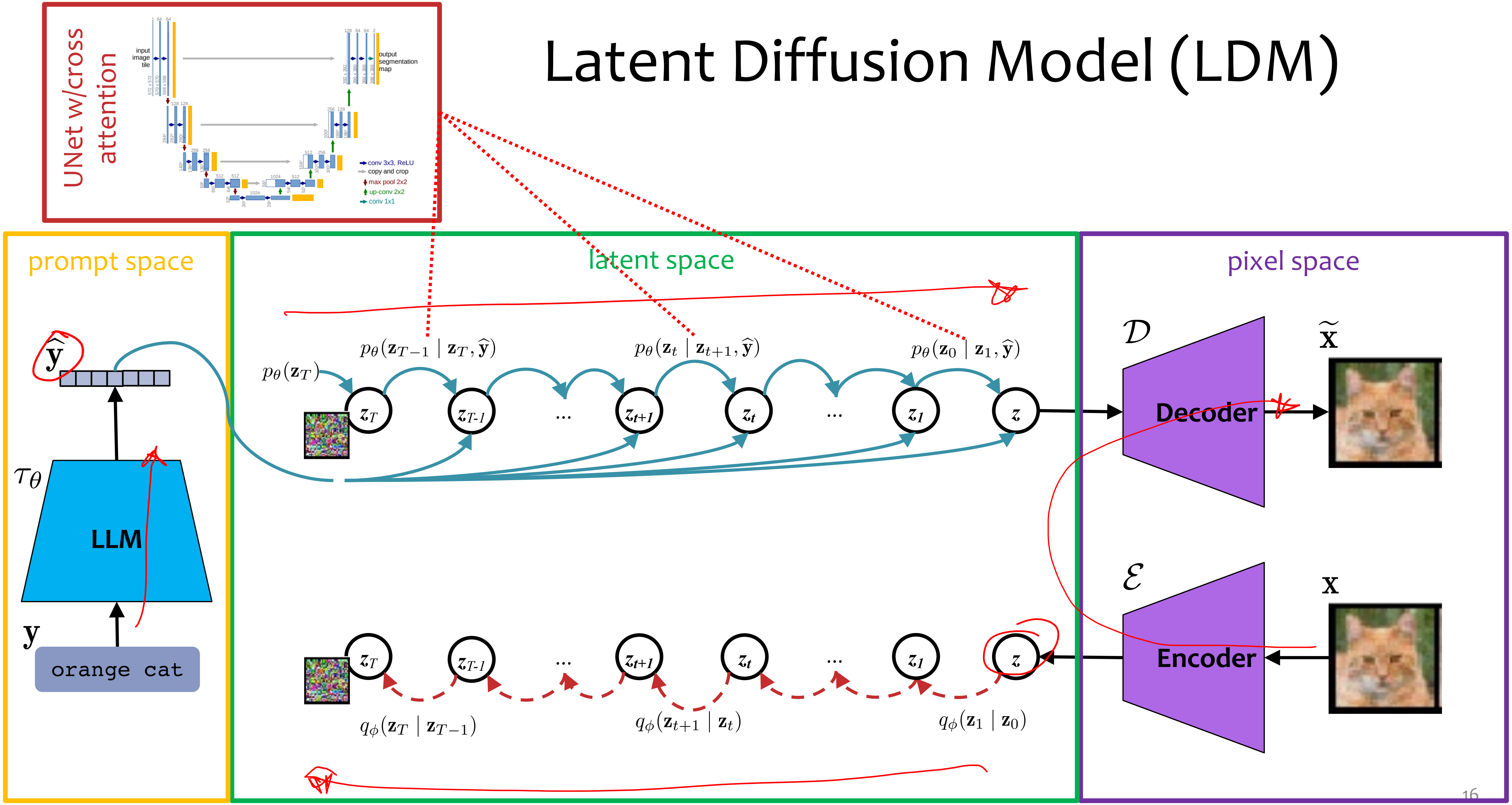
- diffusion models typically operate in pixel space
- yet, training typically takes hundreds of GPU days
  - 150 – 1000 V100 days [Guided Diffusion] (Dhariwal & Nichol, 2021)
  - 256 TPU-v4s for 4 days = 1000 TPU days [Imagen] (Sharia et al., 2022)
- inference is also slow
  - 50k samples in 5 days on A100 GPU [Guided Diffusion] (Dhariwal & Nichol, 2021)
  - 15 seconds per image

Key Idea:

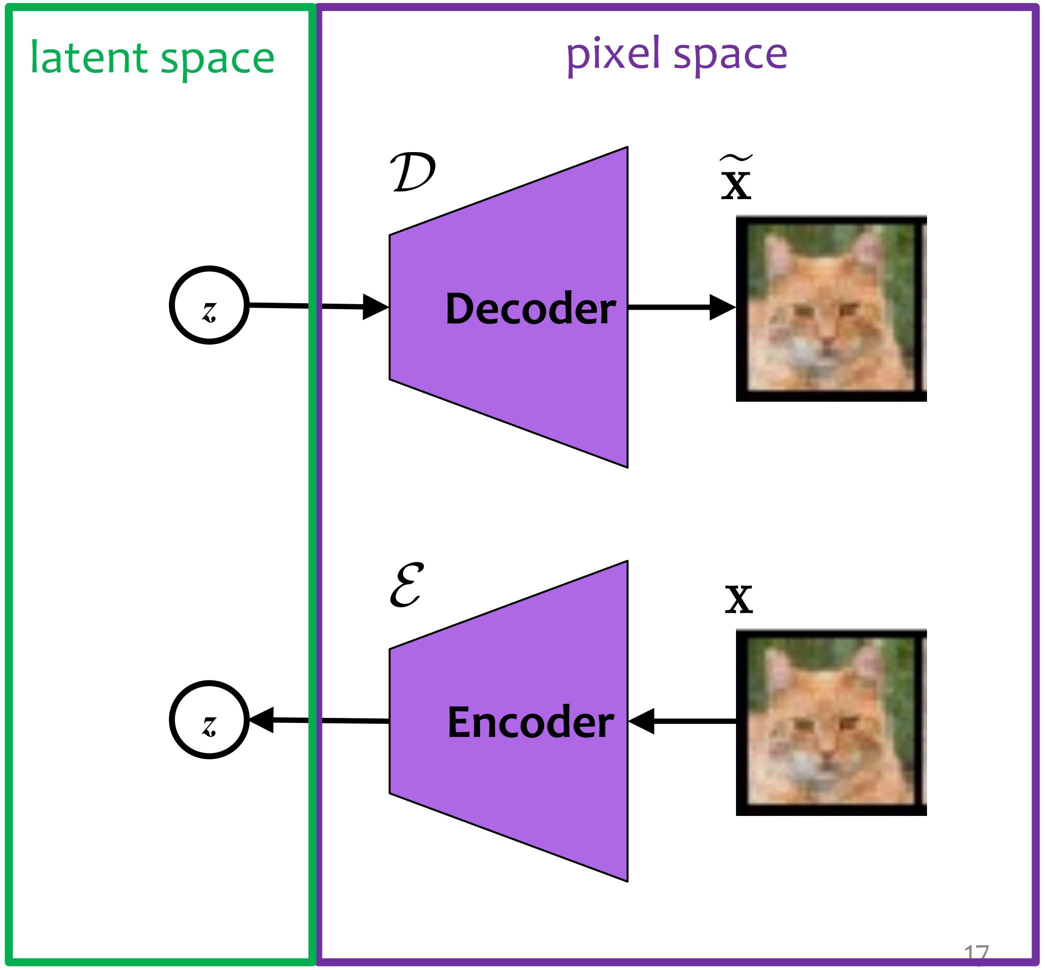
- train an autoencoder (i.e. encoder-decoder model) that learns an efficient latent space that is perceptually equivalent to the data space
- keeping the autoencoder fixed, train a diffusion model on the latent representations of real images z_0 = encoder(x)
  - forward model: latent representation $z_0 \rightarrow$ noise z_T
  - reverse model: noise $z_T \rightarrow$ latent representation z_0
- to generate an image:
  - sample noise z_T
  - apply reverse diffusion model to obtain a latent representation z_0
  - decode the latent representation to an image x
- condition on prompt via cross attention in latent space

Latent Diffusion Model

Latent Diffusion Model (LDM)

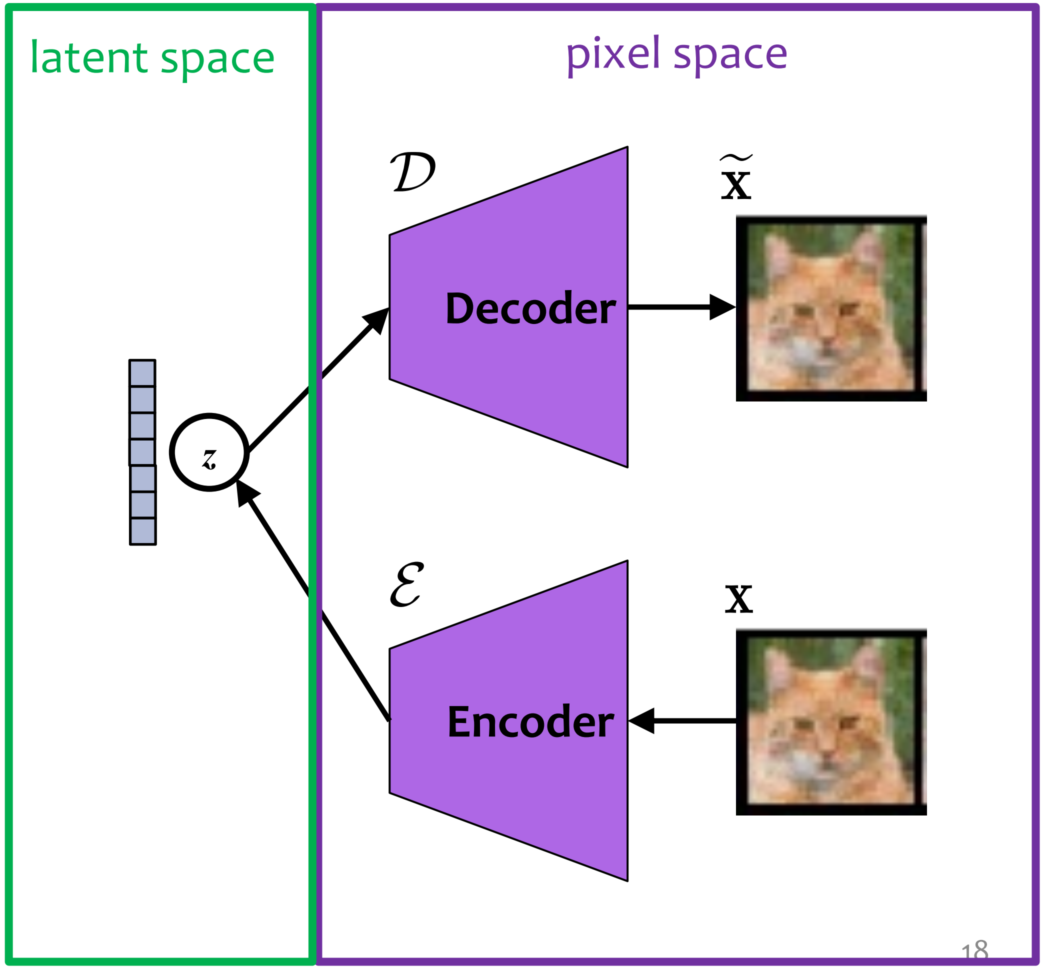


LDM: Autoencoder

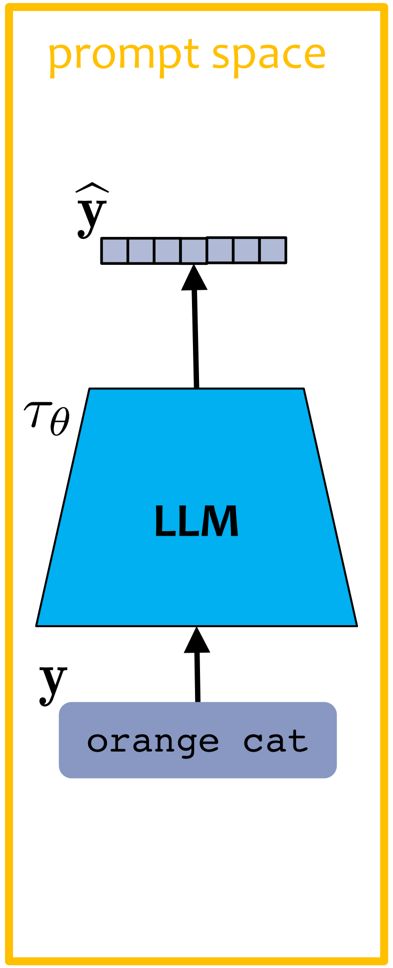


LDM: Autoencoder

- The autoencoder is chosen so that it can project high dimensional images (e.g. 1024x1024) down to low dimensional **latent space** and **faithfully** project back up to **pixel space**
- The original LDM paper considers two options:
  1. a VAE-like model (regularizes the noise towards a Gaussian)
  2. a VQGAN (performs vector quantization in the decoder; i.e., it uses a discrete codebook)
- This model is trained ahead of time just on raw images (no text prompts) and then frozen
- The frozen encoder-decoder can be reused for all subsequent LDM training

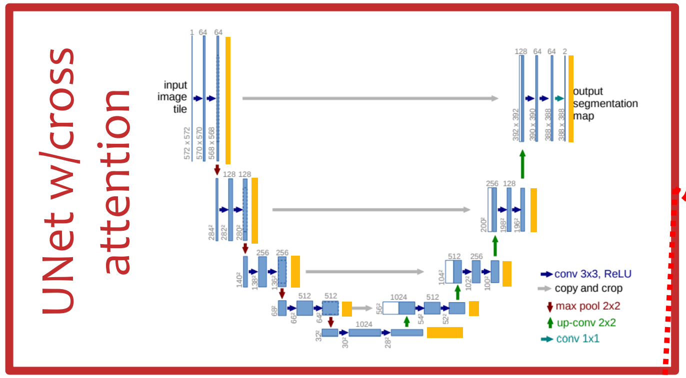


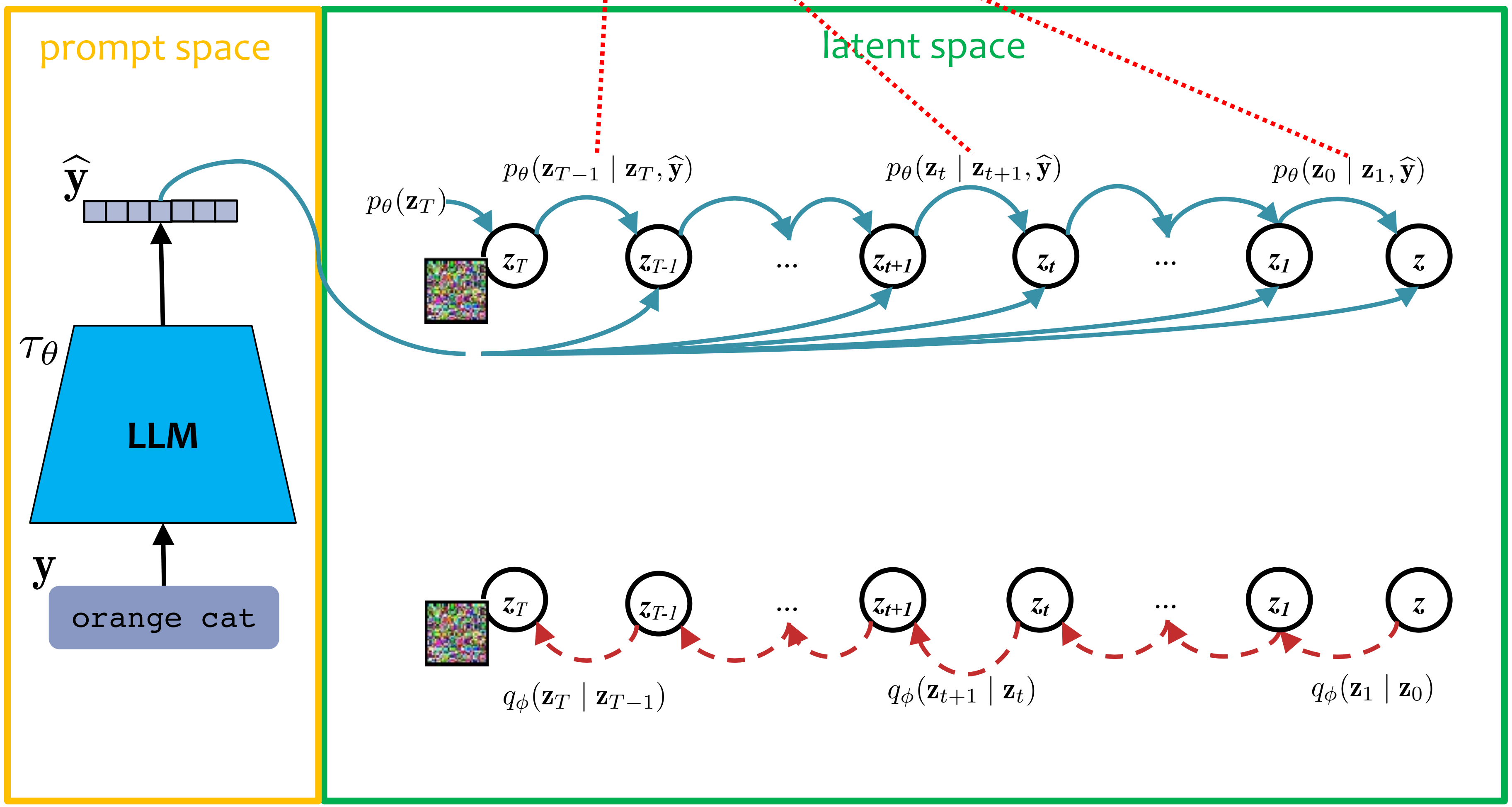
LDM: the Prompt Model



- The prompt model is just a Transformer LM
- We learn its parameters alongside the diffusion model
- The goal is to build up good representations of the text prompts such that they inform the latent diffusion process

LDM: with DDPM





LDM: with DDPM

Noise schedule:

We choose α_t to follow a fixed schedule s.t. $q_\phi(\mathbf{x}_T) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, just like $p_\theta(\mathbf{x}_T)$.

Here we let $\mathbf{z}_0 = \mathbf{z}$, the output of the encoder from our autoencoder

Forward Process:

$$q_\phi(\mathbf{z}_{1:T}) = q(\mathbf{z}_0) \prod_{t=1}^{T} q_\phi(\mathbf{z}_t \mid \mathbf{z}_{t-1})$$

$$q(\mathbf{z}_0) = \text{data distribution}$$
$$q_\phi(\mathbf{z}_t \mid \mathbf{z}_{t-1}) \sim \mathcal{N}(\sqrt{\alpha_t}\mathbf{z}_{t-1}, (1-\alpha_t)\mathbf{I})$$

(Learned) Reverse Process:

$$p_\theta(\mathbf{z}_{1:T}) = p_\theta(\mathbf{z}_T) \prod_{t=1}^{T} p_\theta(\mathbf{z}_{t-1} \mid \mathbf{z}_t, \tau_\theta(y))$$

$$p_\theta(\mathbf{z}_T) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$
$$p_\theta(\mathbf{z}_{t-1} \mid \mathbf{z}_t, \tau_\theta(y)) \sim \mathcal{N}(\mu_\theta(\mathbf{z}_t, t, \tau_\theta(y)), \mathbf{\Sigma}_\theta(\mathbf{z}_t, t))$$

LDM: with DDPM

Noise schedule:

We choose α_t to follow a fixed schedule s.t. $q_\phi(\mathbf{x}_T) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, just like $p_\theta(\mathbf{x}_T)$.

Here we let $\mathbf{z}_0 = \mathbf{z}$, the output of the encoder from our autoencoder

Forward Process:

$$q_\phi(\mathbf{z}_{1:T}) = q(\mathbf{z}_0) \prod_{t=1}^{T} q_\phi(\mathbf{z}_t \mid \mathbf{z}_{t-1})$$

$$q(\mathbf{z}_0) = \text{data}$$

$$q_\phi(\mathbf{z}_t \mid \mathbf{z}_{t-1}) \sim \mathcal{N}(\sqrt{$$

Question: How do we define the mean to condition on the prompt representation?

(Learned) Reverse Process:

$$p_\theta(\mathbf{z}_{1:T}) = p_\theta(\mathbf{z}_T) \prod_{t=1}^{T} p_\theta(\mathbf{z}_{t-1} \mid \mathbf{z}_t, \tau_\theta(y))$$

$$p_\theta(\mathbf{z}_T) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$p_\theta(\mathbf{z}_{t-1} \mid \mathbf{z}_t, \tau_\theta(y)) \sim \mathcal{N}(\mu_\theta(\mathbf{z}_t, t, \tau_\theta(y)), \mathbf{\Sigma}_\theta(\mathbf{z}_t, t))$$

Properties of forward and *exact* reverse process

Recall...

Property #1:

$$q(\mathbf{x}_t \mid \mathbf{x}_0) \sim \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$$

$$\text{where } \bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$$

$\Rightarrow$ we can sample $\mathbf{x}_t$ from $\mathbf{x}_0$ at any timestep t efficiently in closed form

$\Rightarrow$ $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + (1-\bar{\alpha}_t)\boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

Property #2: Estimating $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ is intractable because of its dependence on $q(\mathbf{x}_0)$. However, conditioning on $\mathbf{x}_0$ we can efficiently work with:

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\tilde{\mu}_q(\mathbf{x}_t, \mathbf{x}_0), \sigma_t^2\mathbf{I})$$

$$\text{where } \tilde{\mu}_q(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\bar{\alpha}_t}(1-\alpha_t)}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_t)}{1-\bar{\alpha}_t}\mathbf{x}_t$$

$$= \alpha_t^{(0)}\mathbf{x}_0 + \alpha_t^{(t)}\mathbf{x}_t$$

$$\sigma_t^2 = \frac{(1-\bar{\alpha}_{t-1})(1-\alpha_t)}{1-\bar{\alpha}_t}$$

Property #3: Combining the two previous properties, we can obtain a different parameterization of $\tilde{\mu}_q$ which has been shown empirically to help in learning p_θ.

Rearranging $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + (1-\bar{\alpha}_t)\boldsymbol{\epsilon}$ we have that:

$$\mathbf{x}_0 = (\mathbf{x}_0 + (1-\bar{\alpha}_t)\boldsymbol{\epsilon}) / \sqrt{\bar{\alpha}_t}$$

Substituting this definition of $\mathbf{x}_0$ into property #2's definition of $\tilde{\mu}_q$ gives:

$$\tilde{\mu}_q(\mathbf{x}_t, \mathbf{x}_0) = \alpha_t^{(0)}\mathbf{x}_0 + \alpha_t^{(t)}\mathbf{x}_t$$

$$= \alpha_t^{(0)}\left((\mathbf{x}_0 + (1-\bar{\alpha}_t)\boldsymbol{\epsilon}) / \sqrt{\bar{\alpha}_t}\right) + \alpha_t^{(t)}\mathbf{x}_t$$

$$= \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{(1-\alpha_t)}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}\right)$$

Parameterizing the *learned* reverse process

Recall...

Recall: $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \sim \mathcal{N}(\mu_\theta(\mathbf{x}_t, t), \mathbf{\Sigma}_\theta(\mathbf{x}_t, t))$

Later we will show that given a training sample $\mathbf{x}_0$, we want

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

to be as close as possible to

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$$

Intuitively, this makes sense: if the *learned* reverse process is supposed to subtract away the noise, then whenever we're working with a specific $\mathbf{x}_0$ it should subtract it away exactly as *exact* reverse process would have.

Idea #1: Rather than learn $\Sigma_\theta(\mathbf{x}_t, t)$ just use what we know about $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \sim \mathcal{N}(\tilde{\mu}_q(\mathbf{x}_t, \mathbf{x}_0), \sigma_t^2 \mathbf{I})$:

$$\Sigma_\theta(\mathbf{x}_t, t) = \sigma_t^2 \mathbf{I}$$

Idea #2: Choose μ_θ based on $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$, i.e. we want $\mu_\theta(\mathbf{x}_t, t)$ to be close to $\tilde{\mu}_q(\mathbf{x}_t, \mathbf{x}_0)$. Here are three ways we could parameterize this:

Option C: Learn a network that approximates the $\boldsymbol{\epsilon}$ that gave rise to $\mathbf{x}_t$ from $\mathbf{x}_0$ in the forward process from $\mathbf{x}_t$ and t:

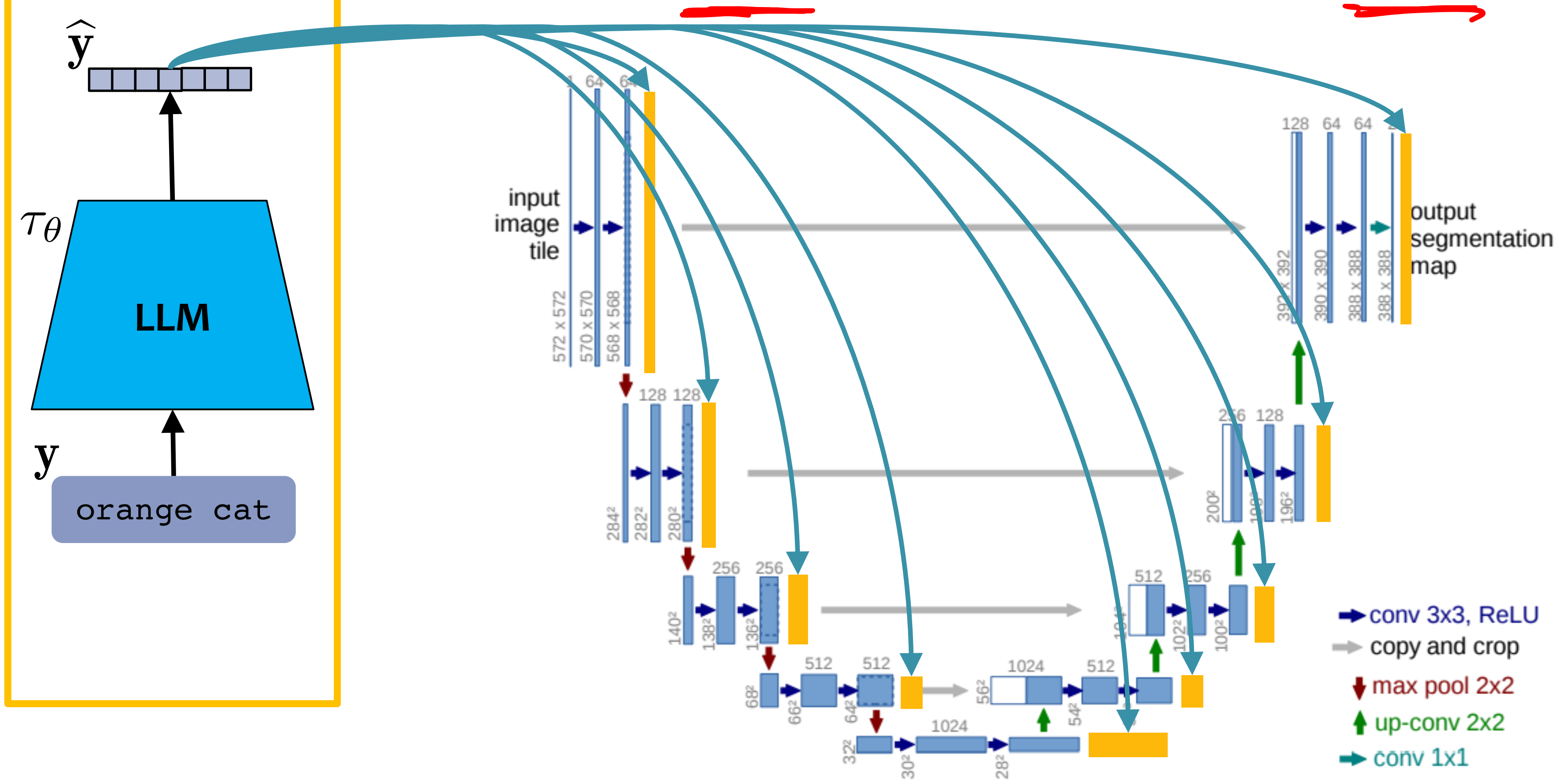
$$\mu_\theta(\mathbf{x}_t, t) = \alpha_t^{(0)} \mathbf{x}_\theta^{(0)}(\mathbf{x}_t, t) + \alpha_t^{(t)} \mathbf{x}_t$$

where $\mathbf{x}_\theta^{(0)}(\mathbf{x}_t, t) = (\mathbf{x}_0 + (1 - \bar{\alpha}_t)\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)) / \sqrt{\bar{\alpha}_t}$

where $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) = \text{UNet}_\theta(\mathbf{x}_t, t)$

LDM: Noise Model

$$\mu_\theta(\mathbf{z}_t, t, \tau_\theta(y)) = \text{UNet}(\mathbf{z}_t, t, \tau_\theta(y))$$



- The noise model includes **cross attention** (yellow boxes) to the representation of the prompt text
- During training we optimize both the parameters of the UNet noise model and the parameters of the LLM simultaneously

LDM: Cross-Attention in Noise Model

- The cross-attention is placed within a larger Transformer layer

Transformer Layer inside UNet

input		$\mathbb{R}^{h\times w\times c}$
LayerNorm		$\mathbb{R}^{h\times w\times c}$
Conv1x1		$\mathbb{R}^{h\times w\times d\cdot n_h}$
Reshape		$\mathbb{R}^{h\cdot w\times d\cdot n_h}$
$\times T$	SelfAttention	$\mathbb{R}^{h\cdot w\times d\cdot n_h}$
	MLP	$\mathbb{R}^{h\cdot w\times d\cdot n_h}$
	CrossAttention	$\mathbb{R}^{h\cdot w\times d\cdot n_h}$
Reshape		$\mathbb{R}^{h\times w\times d\cdot n_h}$
Conv1x1		$\mathbb{R}^{h\times w\times c}$

- The cross-attention modifies the keys and values to be the prompt representation
- The queries are the current layer of UNet

$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)\cdot V$, with

$$Q = W_Q^{(i)}\cdot\varphi_i(z_t),\ K = W_K^{(i)}\cdot\tau_\theta(y),\ V = W_V^{(i)}\cdot\tau_\theta(y).$$

Here, $\varphi_i(z_t)\in\mathbb{R}^{N\times d_\epsilon^i}$ denotes a (flattened) intermediate representation of the UNet implementing ϵ_θ and $W_V^{(i)}\in\mathbb{R}^{d\times d_\epsilon^i}$, $W_Q^{(i)}\in\mathbb{R}^{d\times d_\tau}$ & $W_K^{(i)}\in\mathbb{R}^{d\times d_\tau}$ are learnable projection matrices [36, 97]. See Fig. 3 for a visual depiction.

Figure from http://arxiv.org/abs/2112.10752

LDM: Learning the Diffusion Model + LLM

Given a training sample $\mathbf{z}_0$, we want

$$p_\theta(\mathbf{z}_{t-1} \mid \mathbf{z}_t, \tau_\theta(y))$$

to be as close as possible to

$$q(\mathbf{z}_{t-1} \mid \mathbf{z}_t, \mathbf{z}_0)$$

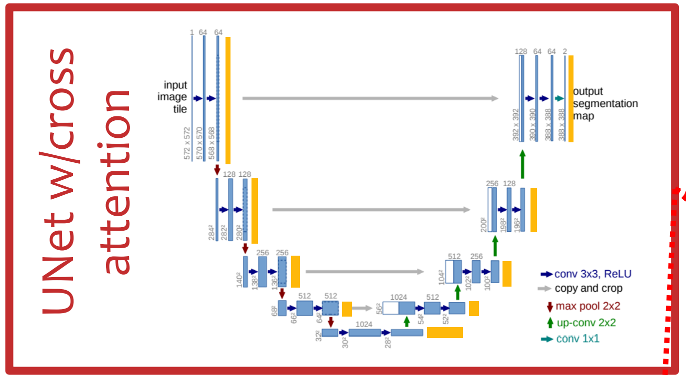
Intuitively, this makes sense: if the *learned* reverse process is supposed to subtract away the noise, then whenever we're working with a specific $\mathbf{z}_0$ it should subtract it away exactly as *exact* reverse process would have.

Objective Function:

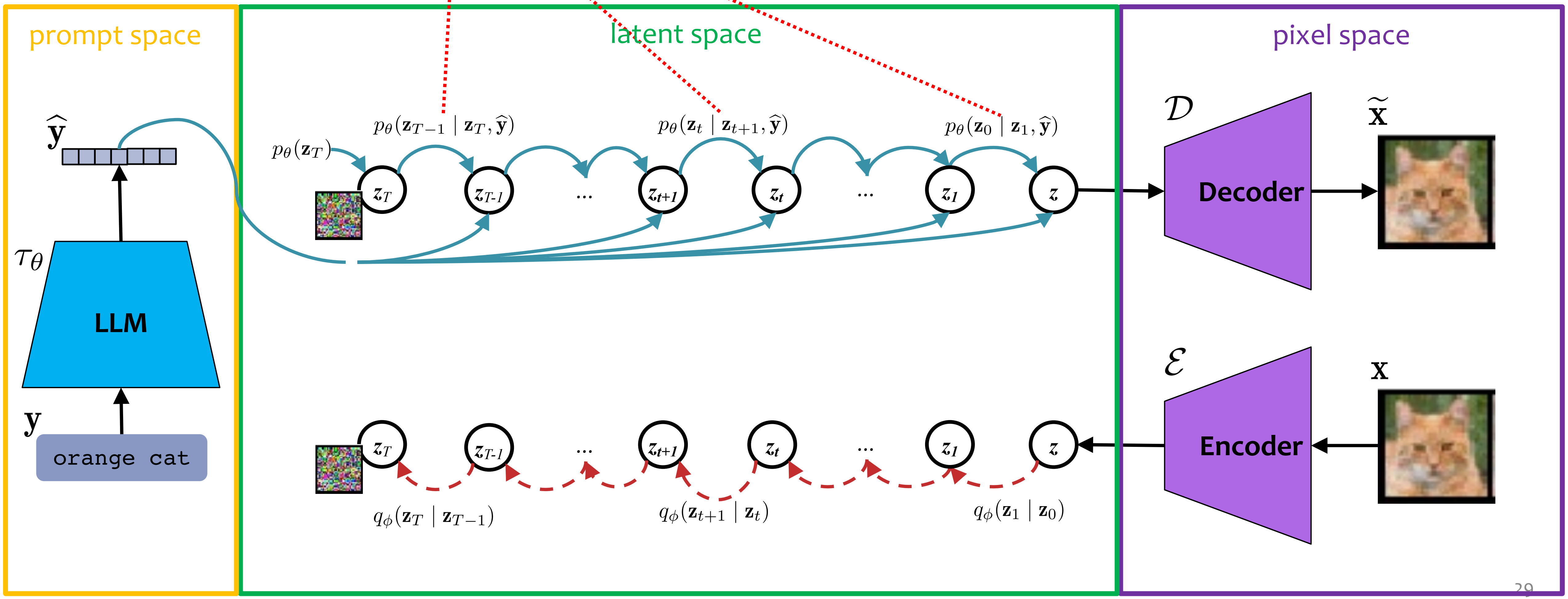
$$L_{LDM} := \mathbb{E}_{\mathcal{E}(x),y,\epsilon\sim\mathcal{N}(0,1),t}\Big[\|\epsilon - \epsilon_\theta(z_t, t, \tau_\theta(y))\|_2^2\Big]$$

Algorithm 1 Training

- initialize θ
- **for** $e \in \{1, \ldots, E\}$ **do**
  - **for** $x_0, y \in \mathcal{D}$ **do**
    - $t \sim \text{Uniform}(1, \ldots, T)$
    - $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    - $\mathbf{x}_t \leftarrow \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}$
    - $\ell_t(\theta) \leftarrow \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \tau_\theta(\mathbf{y}))\|^2$
    - $\theta \leftarrow \theta - \nabla_\theta \ell_t(\theta)$

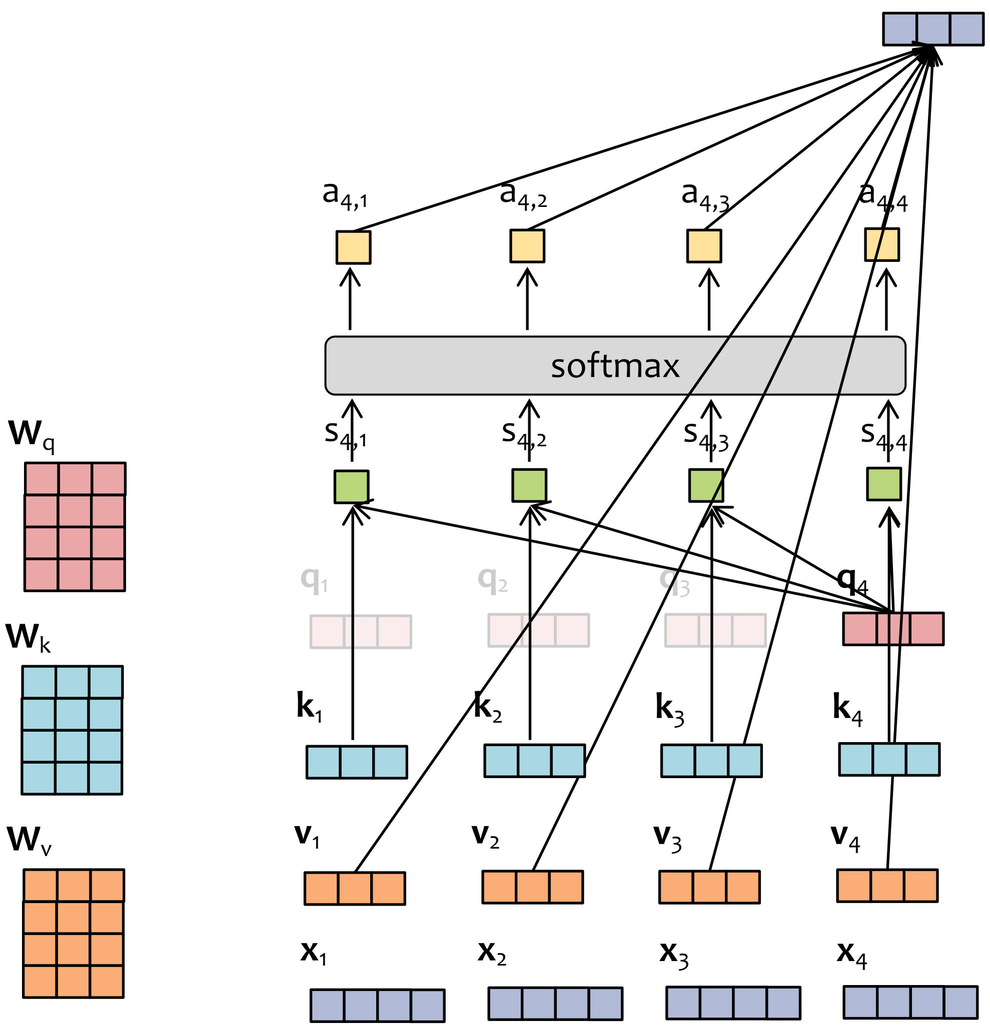


Latent Diffusion Model (LDM)



CROSS-ATTENTION

Scaled Dot-Product Attention



$$\mathbf{x}'_4 = \sum_{j=1}^{4} a_{4,j}\mathbf{v}_j$$

$\mathbf{a}_4 = \text{softmax}(\mathbf{s}_4)$ attention weights

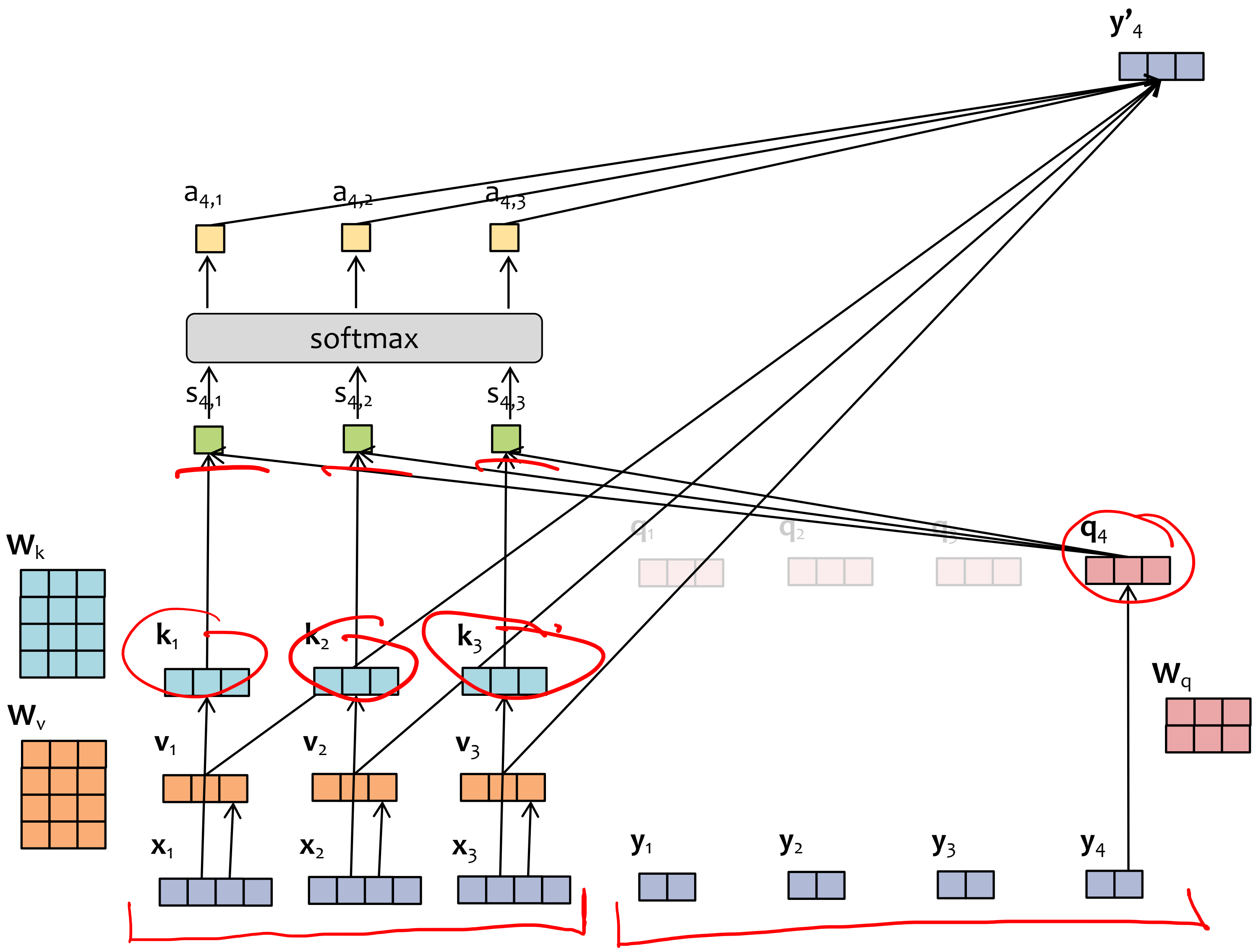
$s_{4,j} = \mathbf{k}_j^T \mathbf{q}_4 / \sqrt{d_k}$ scores

$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$ queries

$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$ keys

$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$ values

Cross Attention



$$\mathbf{y}'_t = \sum_{j=1}^{m} a_{t,j}\mathbf{v}_j, \forall t$$

$$\mathbf{a}_t = \text{softmax}(\mathbf{s}_t), \forall t$$ attention weights

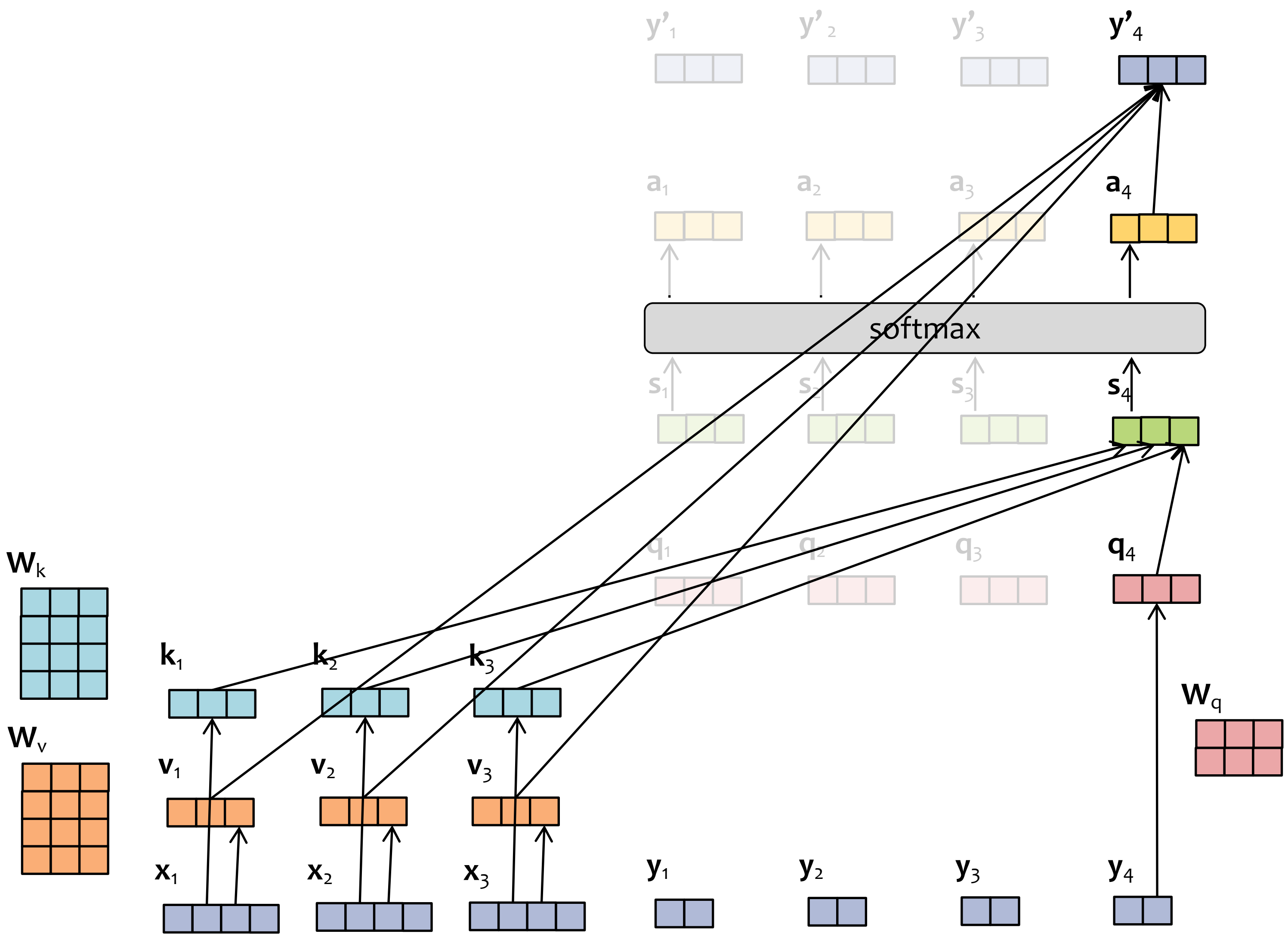
$$s_{t,j} = \mathbf{k}_j^T \mathbf{q}_t / \sqrt{d}, \forall j, t$$ scores

$$\mathbf{q}_t = \mathbf{W}_q^T \mathbf{y}_t, \forall t \in \{1, \ldots, n\}$$ queries

$$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j, \forall j \in \{1, \ldots, m\}$$ keys

$$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j, \forall j \in \{1, \ldots, m\}$$ values

Cross Attention



$$\mathbf{y}'_t = \sum_{j=1}^{m} a_{t,j} \mathbf{v}_j, \forall t$$

$$\mathbf{a}_t = \text{softmax}(\mathbf{s}_t), \forall t$$ attention weights

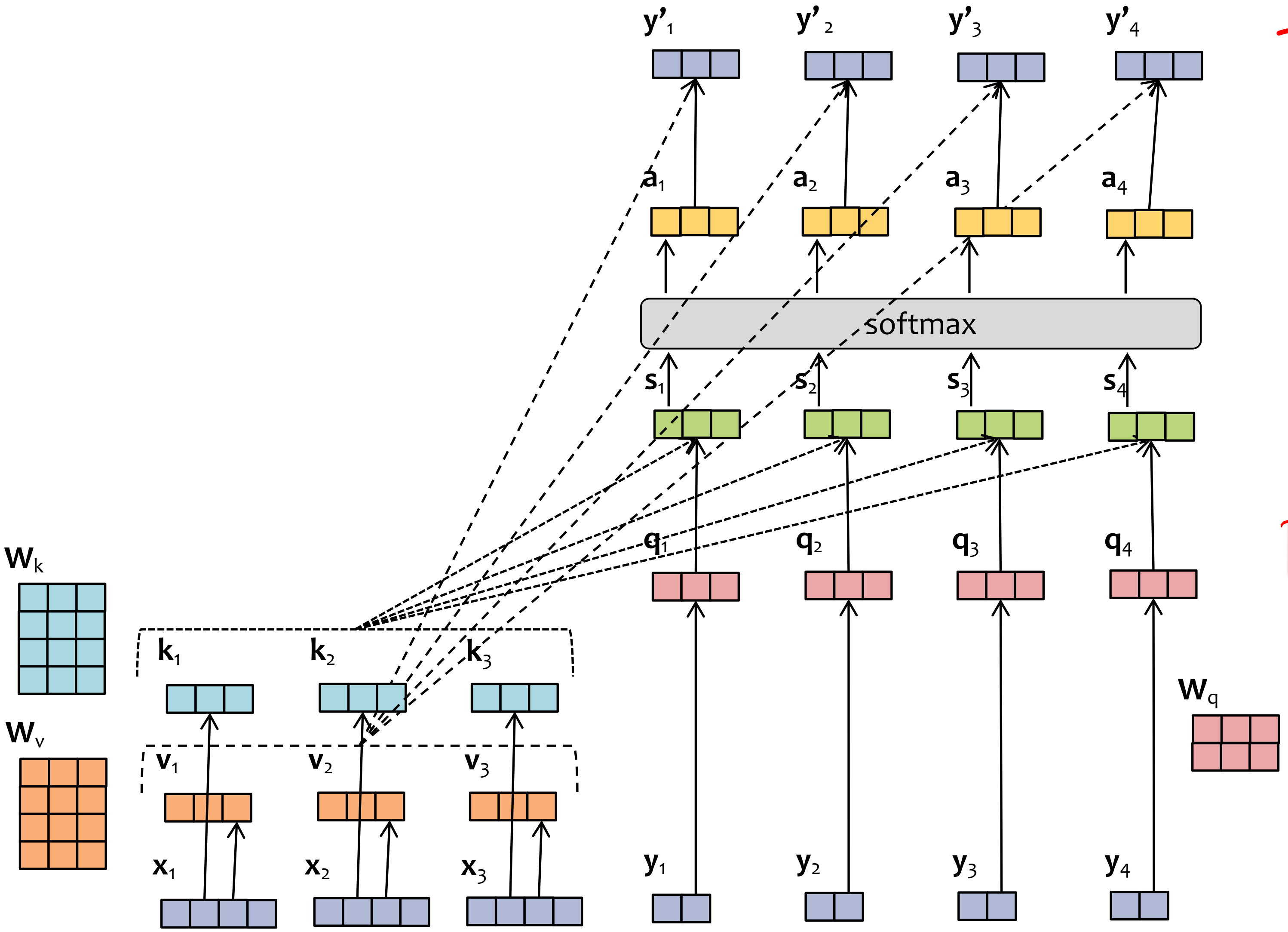
$$s_{t,j} = \mathbf{k}_j^T \mathbf{q}_t / \sqrt{d}, \forall j, t$$ scores

$$\mathbf{q}_t = \mathbf{W}_q^T \mathbf{y}_t, \forall t \in \{1, \ldots, n\}$$ queries

$$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j, \forall j \in \{1, \ldots, m\}$$ keys

$$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j, \forall j \in \{1, \ldots, m\}$$ values

Cross Attention



$$\mathbf{Y}' = \mathbf{AV} = \text{softmax}(\mathbf{QK}^T/\sqrt{d})\mathbf{V}$$

$$\mathbf{A} = \text{softmax}(\mathbf{S})$$ attention weights

$$\mathbf{S} = \mathbf{QK}^T/\sqrt{d} \in \mathbb{R}^{n \times m}$$ scores

$$\mathbf{Q} = \mathbf{YW}_q \in \mathbb{R}^{n \times d}$$ queries

$$\mathbf{K} = \mathbf{XW}_k \in \mathbb{R}^{m \times d}$$ keys

$$\mathbf{V} = \mathbf{XW}_v \in \mathbb{R}^{m \times d}$$ values

PROMPT-TO-PROMPT

Background: Image Editing

- Fixing the Random Seed:
  - A simple baseline for image editing with text: change part of the prompt, **keep the random seed fixed** (e.g. the noise at the start of diffusion), and then run diffusion sampler
  - Problem: the entire **structure** of the image may **change dramatically**
  - Doesn't feel like "editing" at all, more like generation of **unrelated images**

- Mask-based Image Editing:
  - standard approaches to text-based image editing typically **require an image mask** as well
  - the **mask** specifies which part of the image should remain **unchanged**
  - then the **text prompt** informs how the **unmasked part** should be adapted (e.g. by a diffusion model)
  - (Example: Blended Diffusion)

Figure from http://arxiv.org/abs/2208.01626

Figure from http://arxiv.org/abs/2111.14818

Background: Image Editing

- Fixing the Random Seed:
  - A simple baseline for image editing with text: change part of the prompt, **keep the random seed fixed** (e.g. the noise at the start of diffusion), and then run
  - Problem: … e image ma…
  - Doesn't f… ore like … s

the composition is inconsistent in various ways: the background, whole cake vs. single slice, how much cake is in view

- Mask-based Image Editing:
  - standard approaches to text-based image editing typically **require an image mask** as well
  - the **mask** specifies which part of the … main **unchanged**
  - … **ompt** informs how … **art** should be … diffusion model) … ded … ffusion)

here the composition remains consistent across images

Figure from http://arxiv.org/abs/2208.01626

Figure from http://arxiv.org/abs/2111.14818

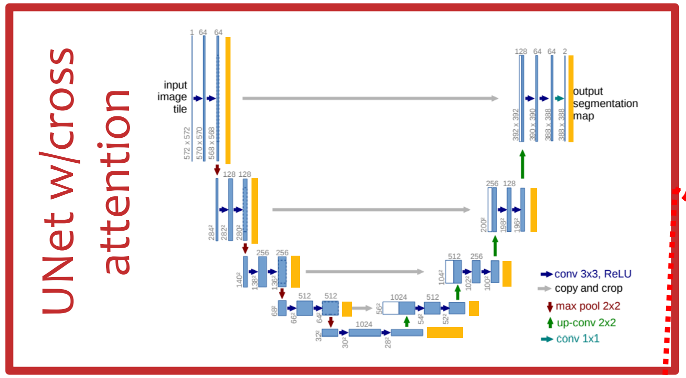
Prompt-to-Prompt

Prompt-to-Prompt:

- **Goal:** edit images with text only and do not require the user to provide a mask
- **Key Idea:**
  - **given** pre-trained latent diffusion model
  - run diffusion model with **original prompt** and **store** the attention weights and cross-attention weights (from the pixels back to the text)
  - re-run diffusion with **edited prompt**, but (carefully) **copy in** the cross-attention weights from the previous run
  - exactly *how* to copy in the attention weights depends on the type of edit
- **Inference only:** no training is involved! we only modify how the samples are drawn from the pre-trained latent diffusion model

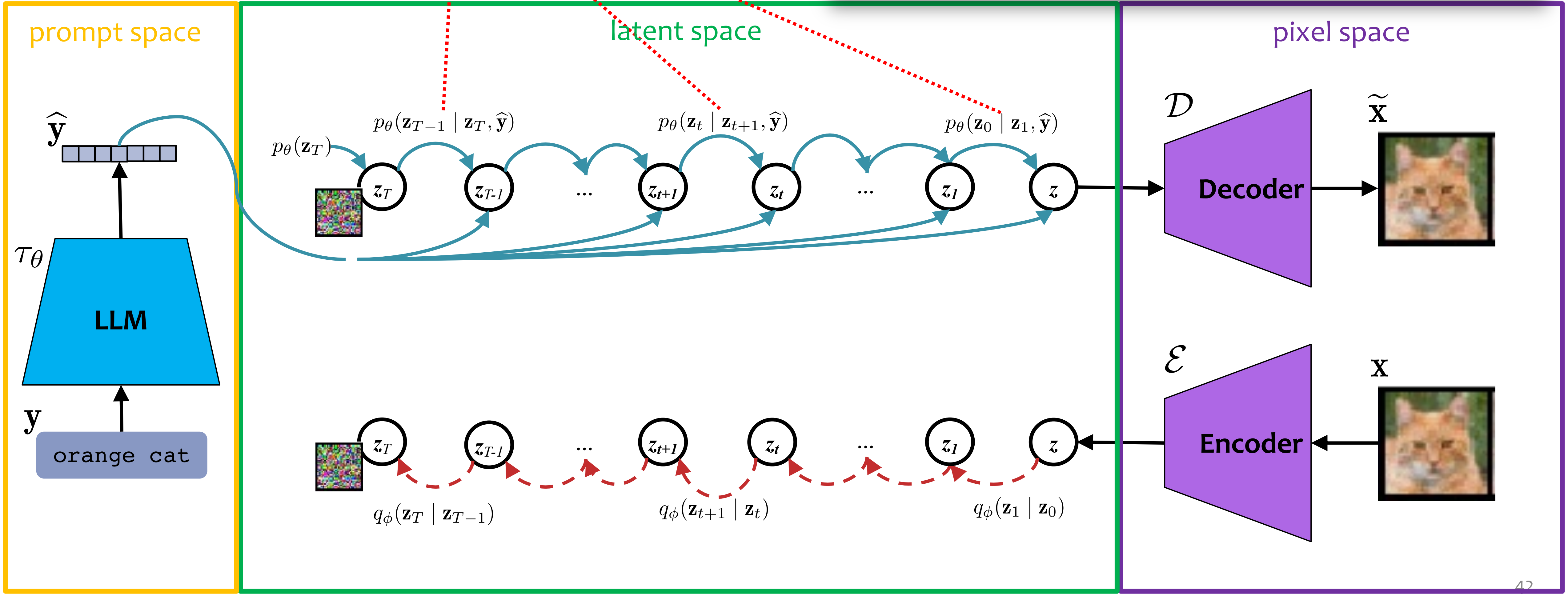
the composition remains consistent across images, but with only the text for guidance (no mask)

Figure from http://arxiv.org/abs/2208.01626



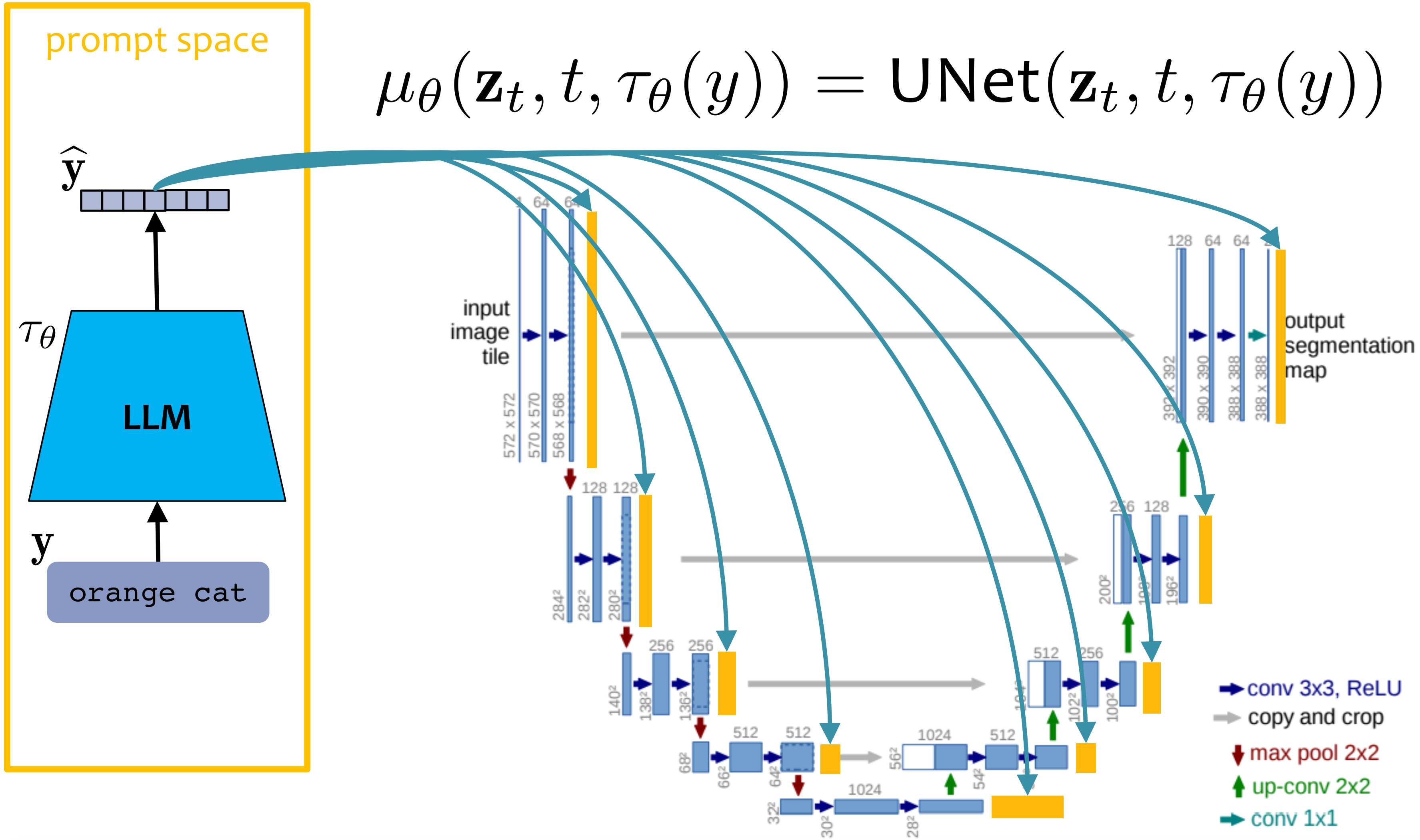
Latent Diffusion Model (LDM)

Prompt-to-prompt assumes we have a pretrained latent diffusion model



LDM: Noise Model

$$\mu_\theta(\mathbf{z}_t, t, \tau_\theta(y)) = \text{UNet}(\mathbf{z}_t, t, \tau_\theta(y))$$



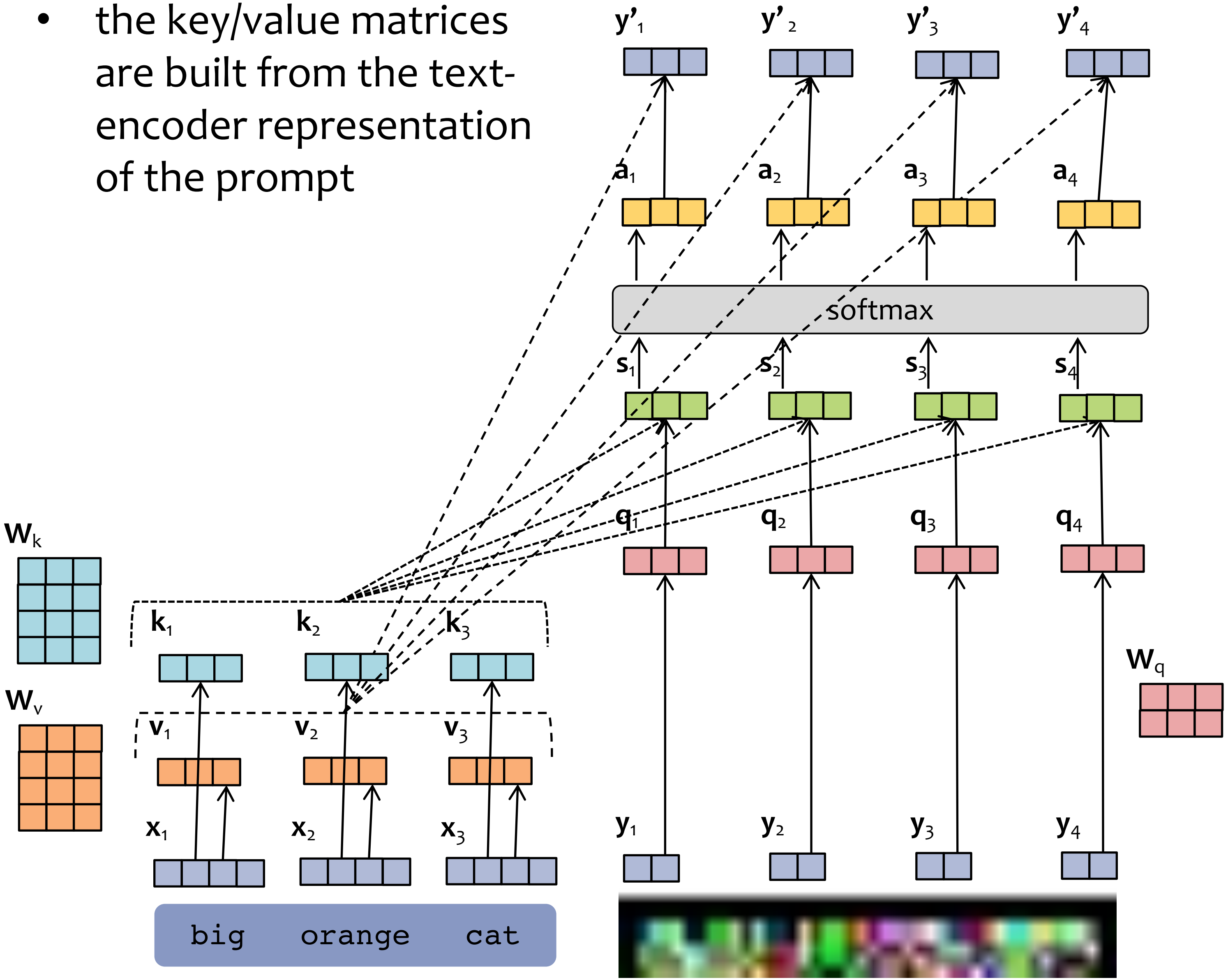
- The noise model includes **cross attention** (yellow boxes) to the representation of the prompt text
- During training we optimize both the parameters of the UNet noise model and the parameters of the LLM simultaneously

Prompt-to-prompt modifies the cross attention in LDM, which looks at both the text encoding and the (latent) representation of the image

Cross-Attention in LDM:

- the query matrix is built from a layer of UNet
- the key/value matrices are built from the text-encoder representation of the prompt

LDM: Cross-Attention



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

$$\mathbf{A} = \text{softmax}(\mathbf{S})$$

$$\mathbf{S} = \mathbf{Q}\mathbf{K}^T/\sqrt{d} \in \mathbb{R}^{n \times m}$$

$$\mathbf{Q} = \mathbf{Y}\mathbf{W}_q \in \mathbb{R}^{n \times d}$$

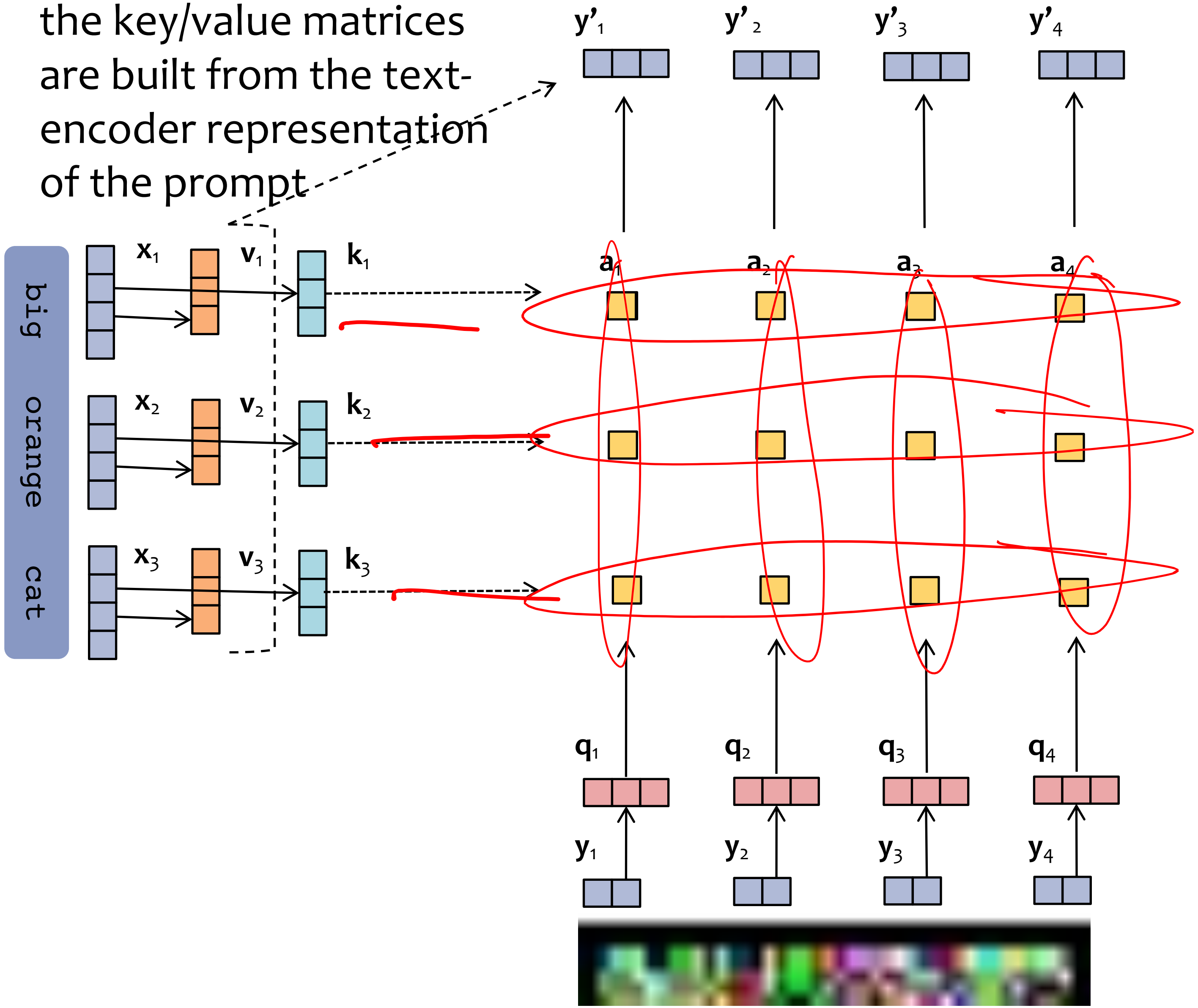
$$\mathbf{K} = \mathbf{X}\mathbf{W}_k \in \mathbb{R}^{m \times d}$$

$$\mathbf{V} = \mathbf{X}\mathbf{W}_v \in \mathbb{R}^{m \times d}$$

Cross-Attention in LDM:

- the query matrix is built from a layer of UNet
- the key/value matrices are built from the text-encoder representation of the prompt

LDM: Cross-Attention



$$\mathbf{Y} = \mathbf{AV} = \text{softmax}(\mathbf{QK}^T/\sqrt{d})\mathbf{V}$$

$$\mathbf{A} = \text{softmax}(\mathbf{S})$$ (attention weights)

$$\mathbf{S} = \mathbf{QK}^T/\sqrt{d} \in \mathbb{R}^{n \times m}$$

$$\mathbf{Q} = \mathbf{YW}_q \in \mathbb{R}^{n \times d}$$

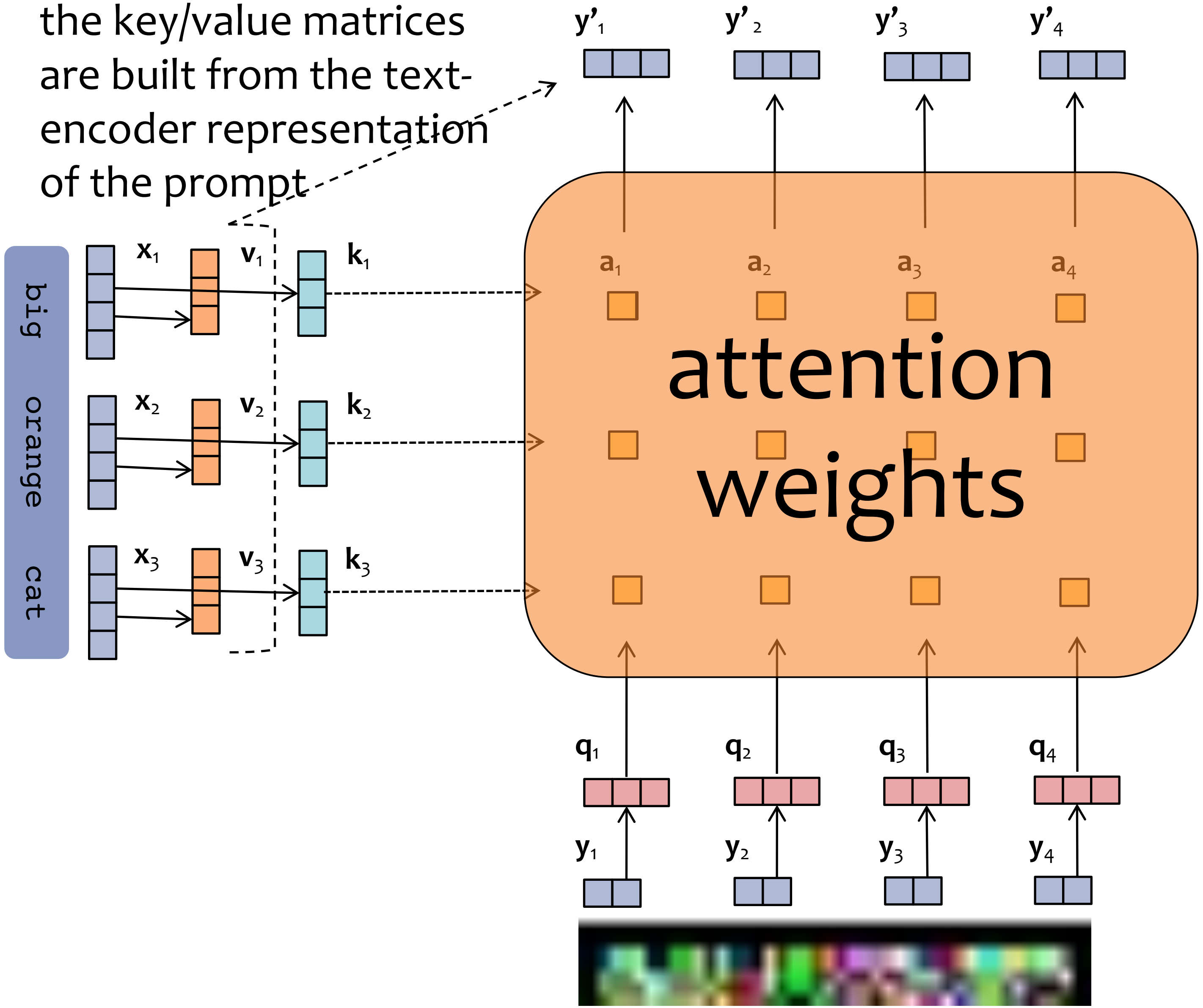
$$\mathbf{K} = \mathbf{XW}_k \in \mathbb{R}^{m \times d}$$

$$\mathbf{V} = \mathbf{XW}_v \in \mathbb{R}^{m \times d}$$

Cross-Attention in LDM:

- the query matrix is built from a layer of UNet
- the key/value matrices are built from the text-encoder representation of the prompt

LDM: Cross-Attention



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

$$\mathbf{A} = \text{softmax}(\mathbf{S}) \qquad \text{(attention weights)}$$

$$\mathbf{S} = \mathbf{Q}\mathbf{K}^T/\sqrt{d} \in \mathbb{R}^{n \times m}$$

$$\mathbf{Q} = \mathbf{Y}\mathbf{W}_q \in \mathbb{R}^{n \times d}$$

$$\mathbf{K} = \mathbf{X}\mathbf{W}_k \in \mathbb{R}^{m \times d}$$

$$\mathbf{V} = \mathbf{X}\mathbf{W}_v \in \mathbb{R}^{m \times d}$$

The actual attention and cross-attention blocks are **multi-head**

Prompt-to-Prompt: Editing Cross Attention

notation change: rep of prompt

Prompt-to-Prompt:

- **Goal:** edit images with text only and do not require the user to provide a mask
- **Key Idea:**
  - **given** pre-trained latent diffusion model
  - run diffusion model with **original prompt** and **store** the attention weights and cross-attention weights (from the pixels back to the text)
  - re-run diffusion with **edited prompt**, but (carefully) **copy in** the cross-attention weights from the previous run
  - exactly *how* to copy in the attention weights depends on the type of edit
- **Inference only:** no training is involved! we only modify how the samples are drawn from the pre-trained latent diffusion model

Algorithm 1 Prompt-to-Prompt image editing

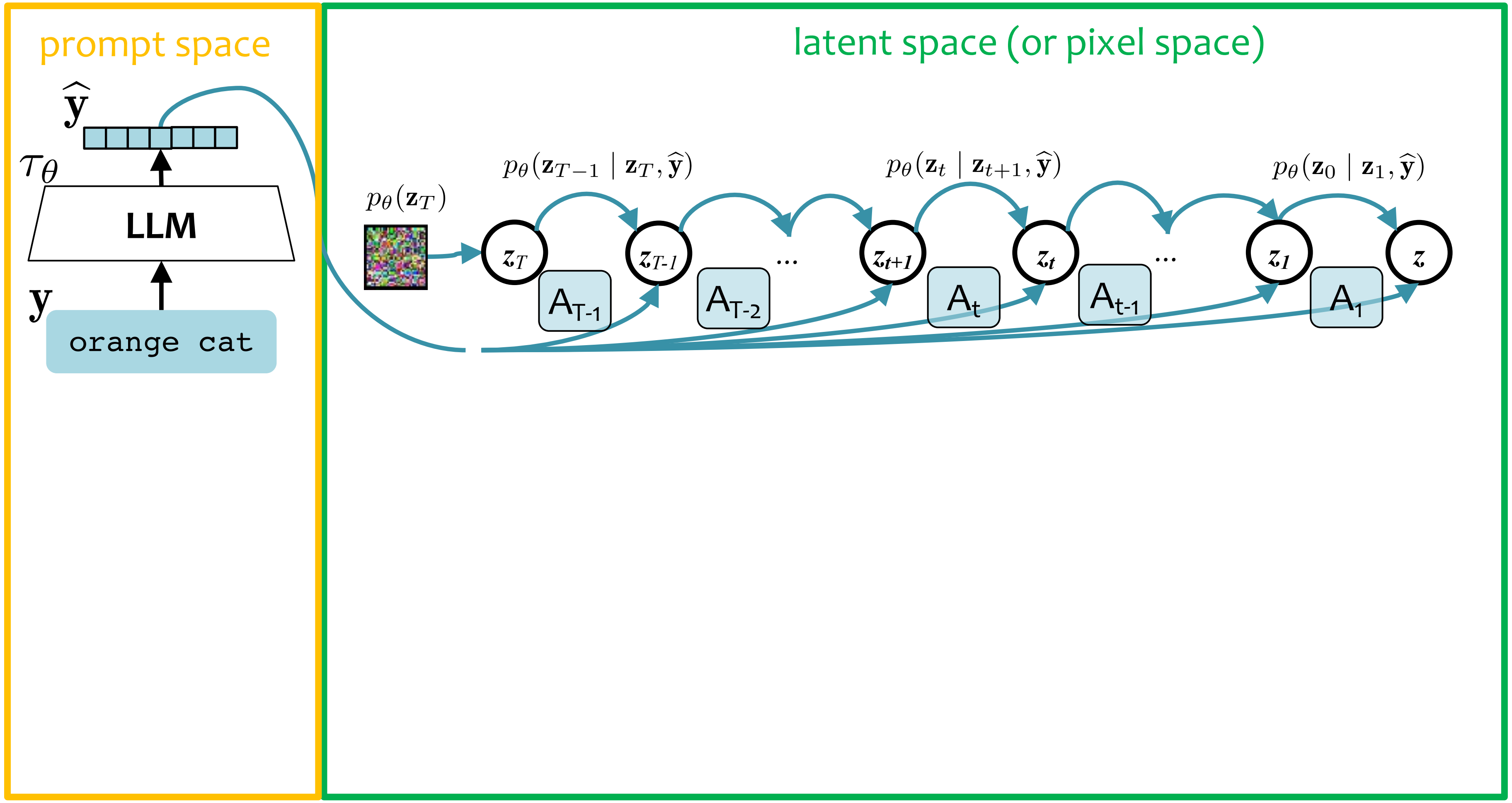
Input: A source prompt $\mathbf{y}$, a target prompt $\mathbf{y}^*$, and a random seed s.
Output: A source image x_{src} and an edited image x_{dst}.
$\mathbf{z}_T \sim \mathcal{N}(0, I)$ a unit Gaussian random variable with random seed s;
$\mathbf{z}_T^* \leftarrow \mathbf{z}_T$;
for $t = T, T-1, \ldots, 1$ **do**
    $\mathbf{z}_{t-1}, \mathbf{A}_t \leftarrow DM(\mathbf{z}_t, \mathbf{y}, t, s)$;
    $\mathbf{A}_t^* \leftarrow DM(\mathbf{z}_t^*, \mathbf{y}^*, t, s)$;
    $\widehat{\mathbf{A}}_t \leftarrow \text{Edit}(\mathbf{A}_t, \mathbf{A}_t^*, t)$;
    $\mathbf{z}_{t-1}^* \leftarrow DM(\mathbf{z}_t^*, \mathbf{y}^*, t, s)\{\mathbf{A} \leftarrow \widehat{\mathbf{A}}_t\}$;
return $(\mathbf{z}_0, \mathbf{z}_0^*)$

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

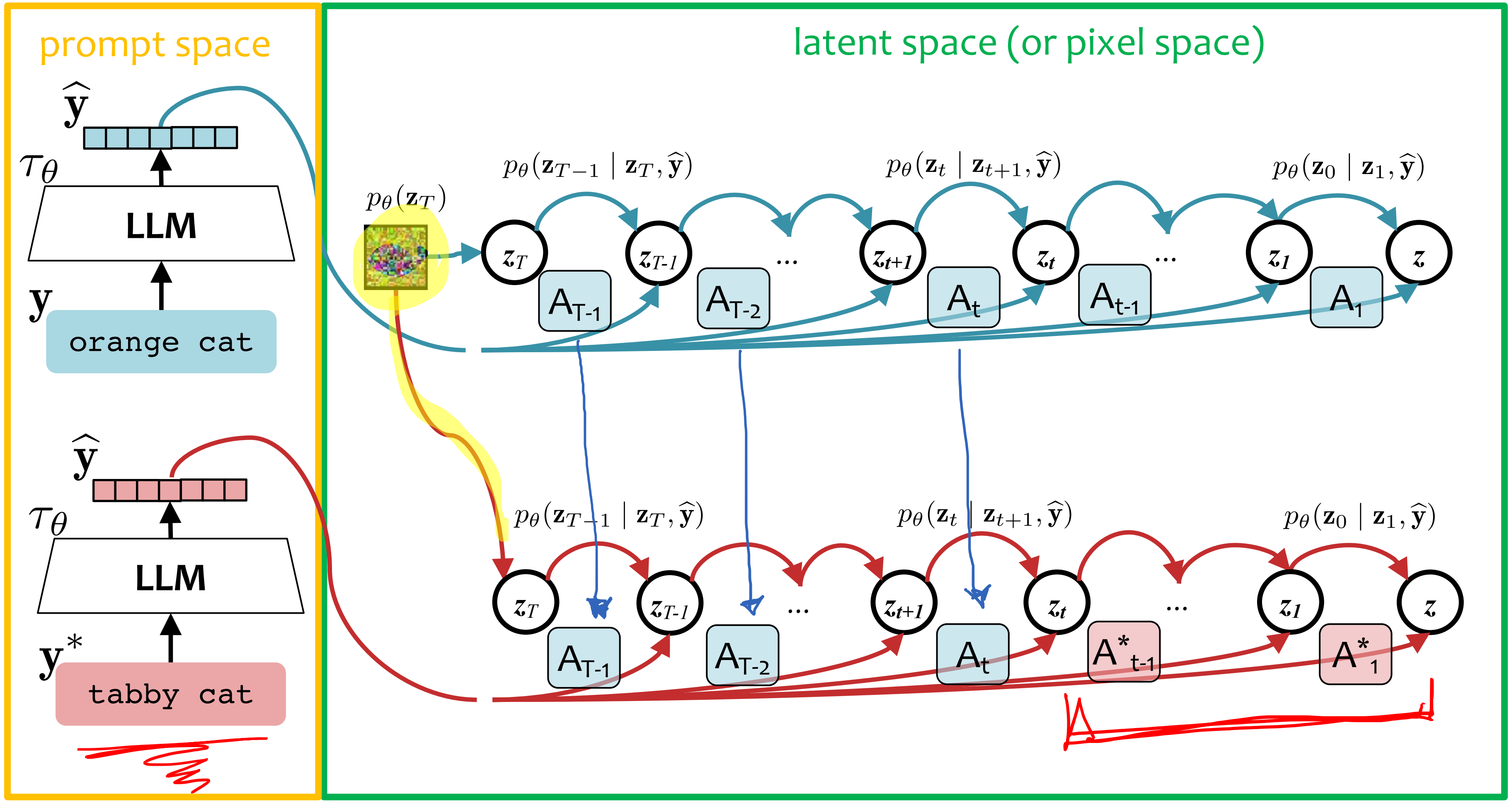
Figure from http://arxiv.org/abs/2208.01626

Prompt-to-Prompt: Editing Cross Attention

1. encode the original prompt y
2. run diffusion on y and obtain attention weights $A_{T-1},\ldots,A_1$



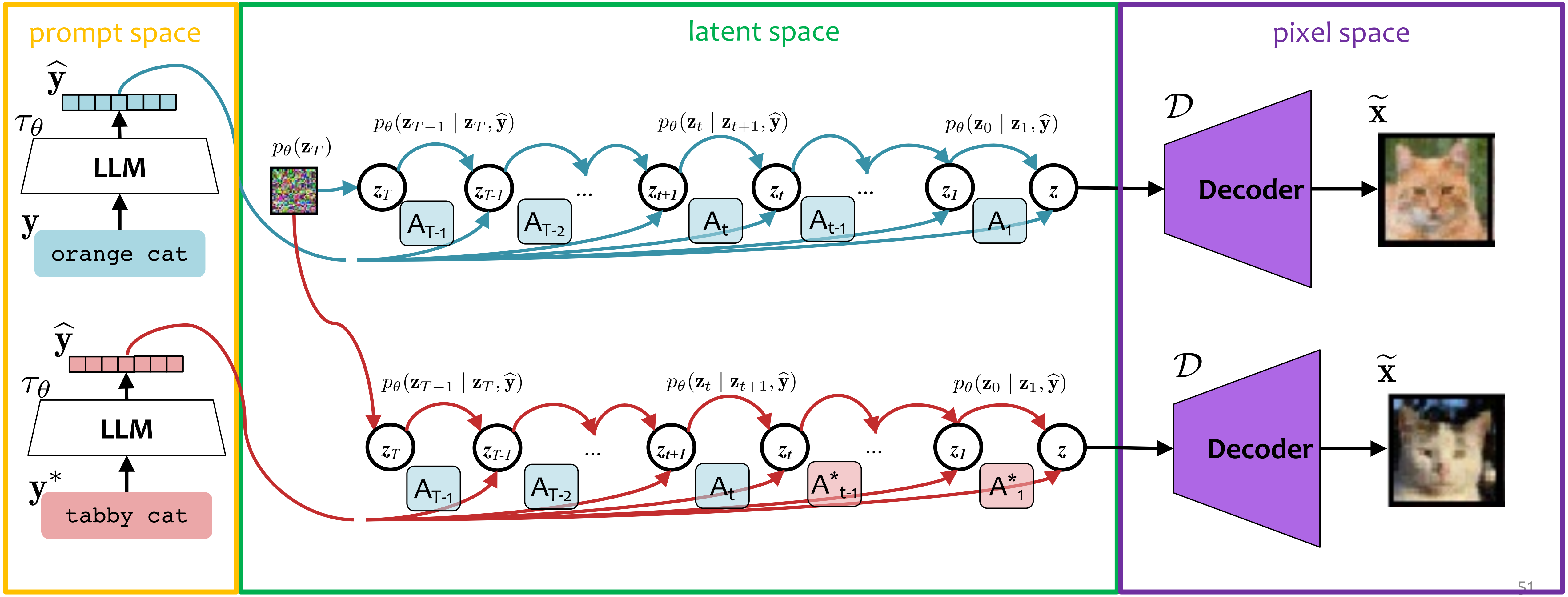
Prompt-to-Prompt: Editing Cross Attention



1. encode the original prompt y
2. run diffusion on y and obtain attention weights $A_{T-1}, \ldots, A_1$
3. encode the modified prompt y*
4. run diffusion again
   a) reuse the noise z_T from the original run
   b) use the attention weights from the original run until timestep τ $A_{T-1}, \ldots, A_t$
   c) then switch to using attention weights from this current run $A^*_{t-1}, \ldots, A^*_1$
   d) regardless of which attention weights, you still attend to y*

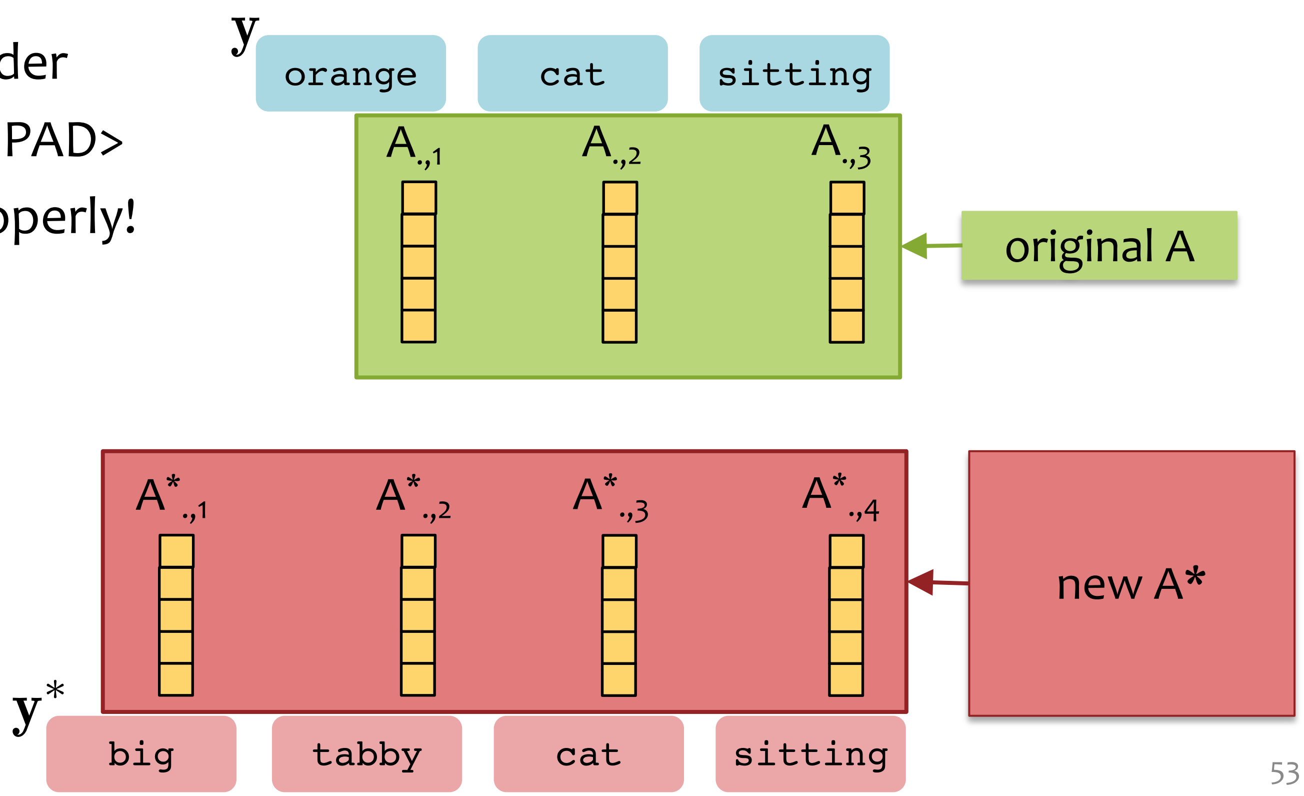
Prompt-to-Prompt: Editing Cross Attention

5. if running in latent space, then use decoder to recover pixel space representation



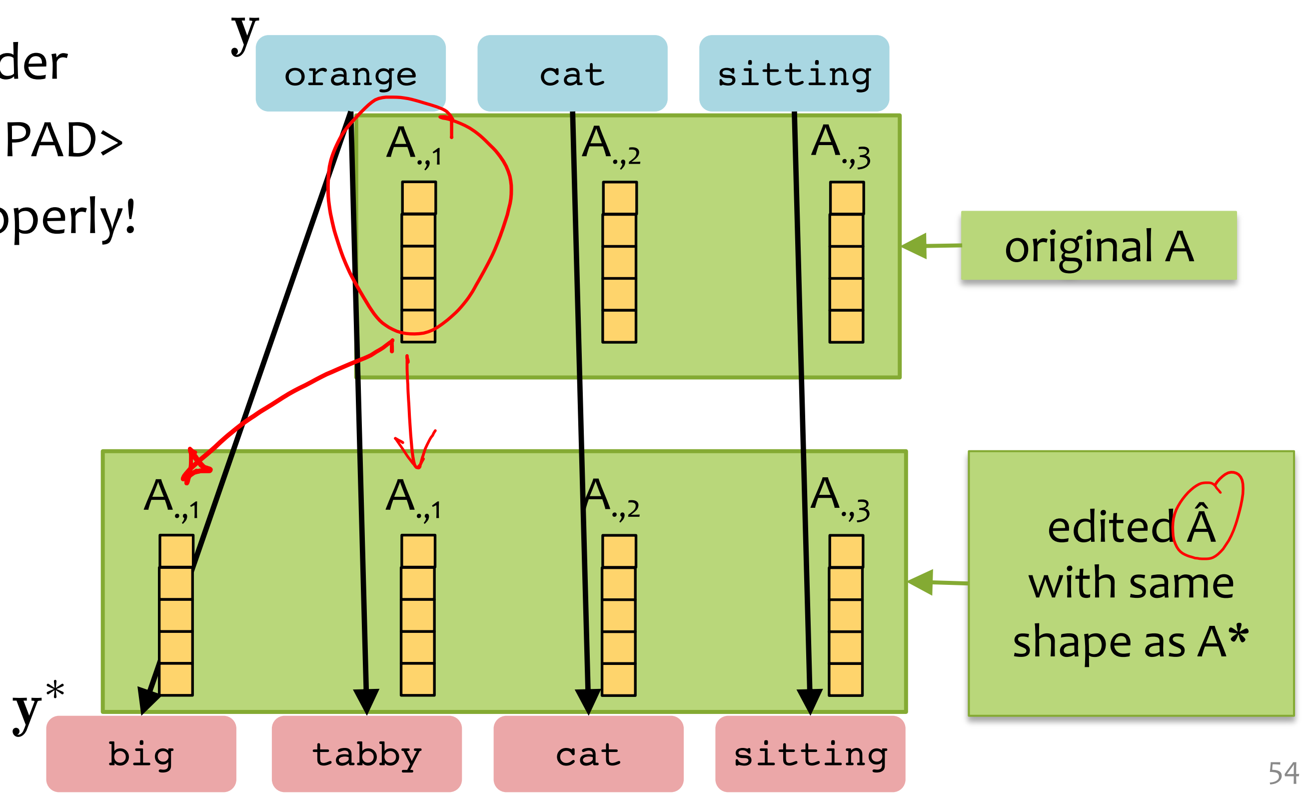
Attention Swapping

- Problem: What if A_t and A^*_t are not the same shape?
- Solution: Swap in just the appropriate parts!
  - The dimension in latent space will always remain constant (e.g. 1024)
  - The dimension in text prompt space also remain constant if we use a fixed length encoder
    - e.g. length = 77, if we use CLIP encoder
    - orange cat sitting <PAD> <PAD> … <PAD>
  - However, the words might not align properly!
- Example:
  - we replace "orange" with "big tabby"
  - then copy the attention weights for "orange" to both "big" and "tabby" in the new attention weights



Attention Swapping

- Problem: What if A_t and A^*_t are not the same shape?
- Solution: Swap in just the appropriate parts!
  - The dimension in latent space will always remain constant (e.g. 1024)
  - The dimension in text prompt space also remain constant if we use a fixed length encoder
    - e.g. length = 77, if we use CLIP encoder
    - orange cat sitting <PAD> <PAD> … <PAD>
  - However, the words might not align properly!
- Example:
  - we replace "orange" with "big tabby"
  - then copy the attention weights for "orange" to both "big" and "tabby" in the new attention weights



Attention Swapping

- We need to do this swapping for every batch, for every head, and for every timestep (until tau)
- Each row corresponds to a different latent space dimension
- Efficiency trick: define a mapper matrix M such that Â = AM

Â

1	1	6	11
2	2	7	12
3	3	8	13
4	4	9	14
5	5	10	15

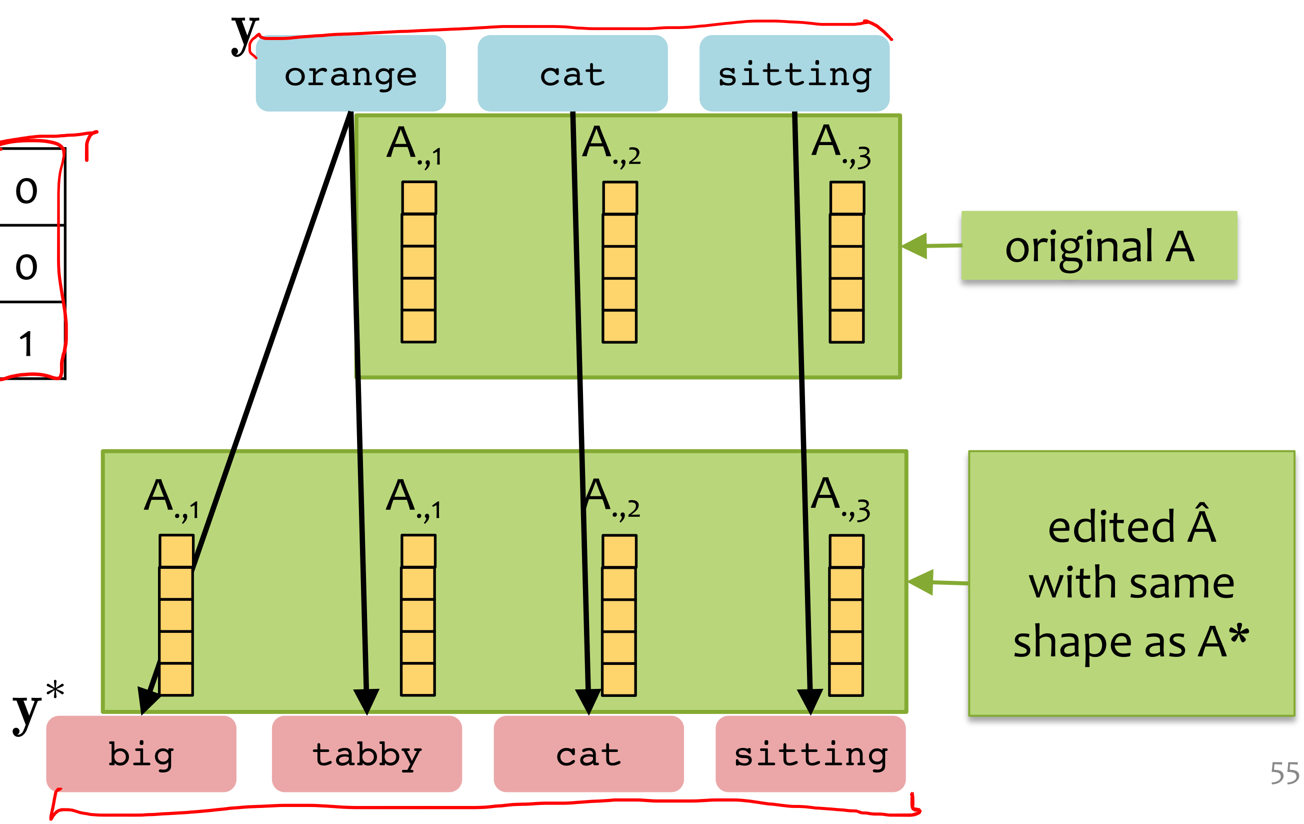
=

A

1	6	11
2	7	12
3	8	13
4	9	14
5	10	15

M

1	1	0	0
0	0	1	0
0	0	0	1



Attention Swapping

- We need to do this swapping for every batch, for every head, and for every timestep (until tau)
- Each row corresponds to a different latent space dimension
- Efficiency trick: define a mapper matrix M such that $\hat{A} = AM$
- Instead of copying, we average over “big” and “tabby”

$\hat{A}$

1/2	1/2	6	11
2/2	2/2	7	12
3/2	3/2	8	13
4/2	4/2	9	14
5/2	5/2	10	15

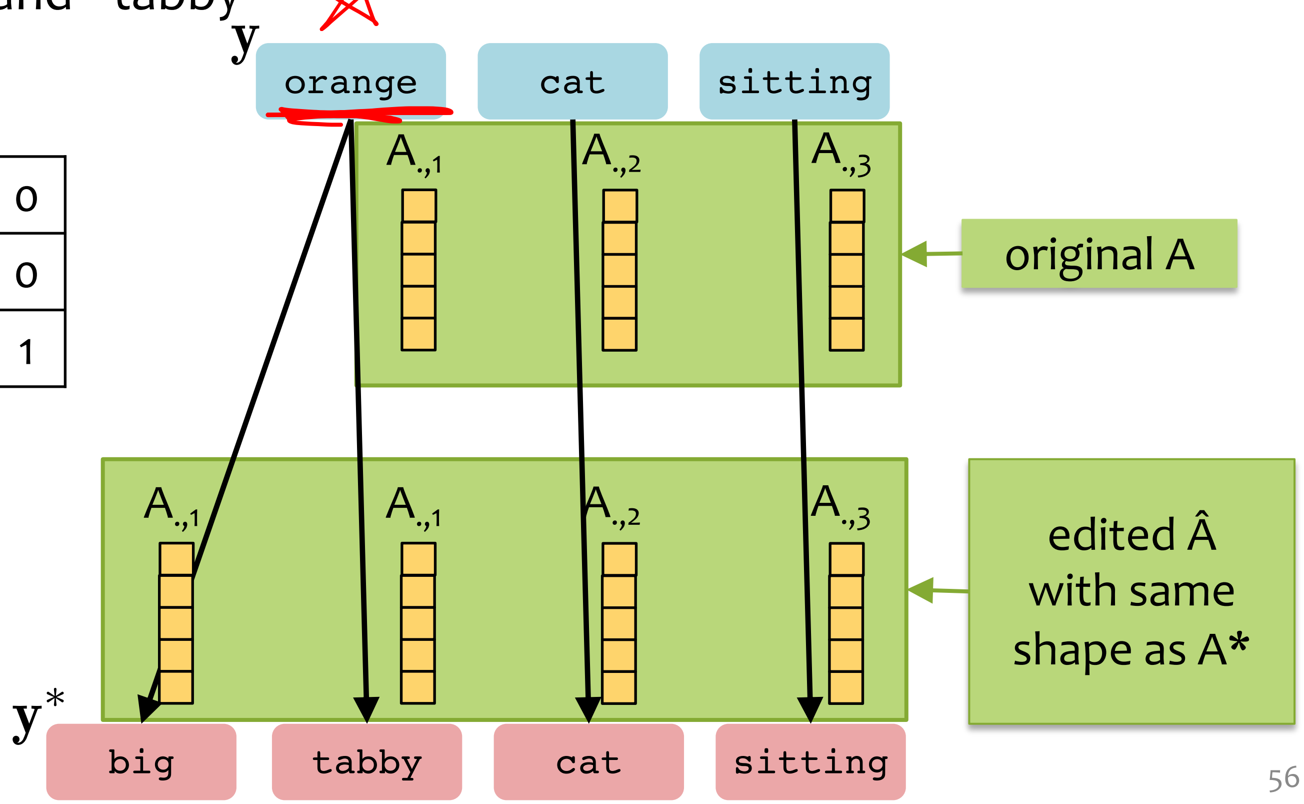
=

A

1	6	11
2	7	12
3	8	13
4	9	14
5	10	15

M

.5	.5	0	0
0	0	1	0
0	0	0	1



Attention Swapping

- We need to do this swapping for every batch, for every head, and for every timestep (until tau)
- Each row corresponds to a different latent space dimension
- Efficiency trick: define a mapper matrix M such that Â = AM
- Instead of copying, we average over "big" and "orange"

Â

(1+6)/2	6	11
(2+7)/2	7	12
(3+8)/2	8	13
(4+9)/2	9	14
(5+10)/2	10	15

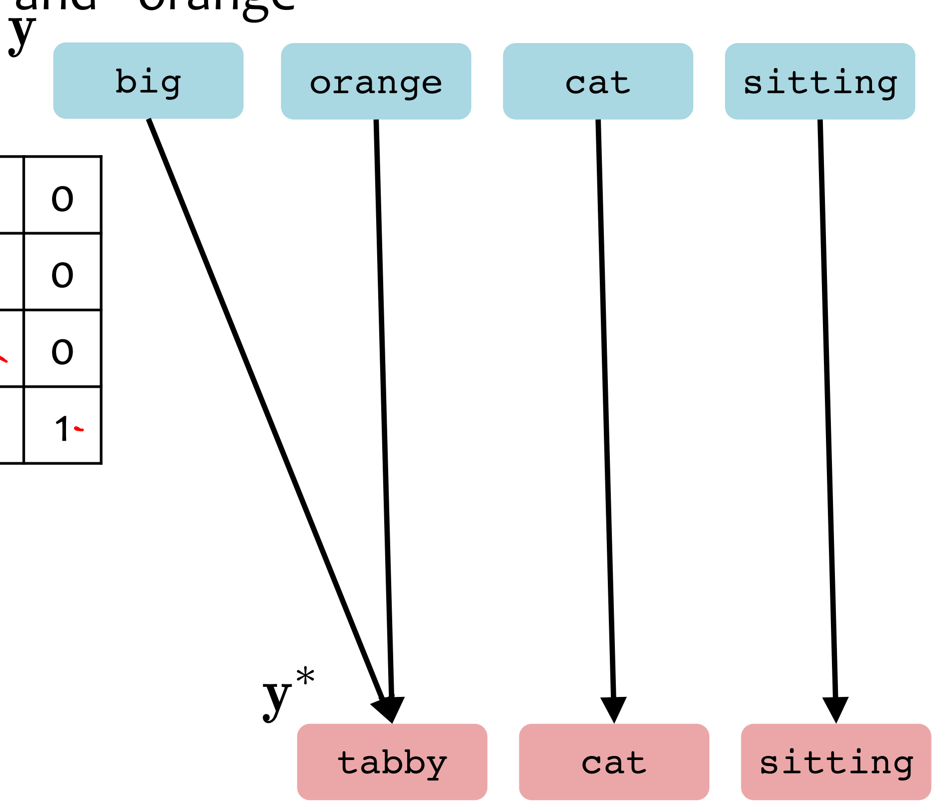
=

A

1	6	11	16
2	7	12	17
3	8	13	18
4	9	14	19
5	10	15	20

M

.5	0	0
.5	0	0
0	1	0
0	0	1



Attention Swapping

- We need to do this swapping for every batch, for every head, and for every timestep (until tau)
- Each row corresponds to a different latent space dimension
- Efficiency trick: define a mapper matrix M such that Â = AM

Â

1	6	11
2	7	12
3	8	13
4	9	14
5	10	15

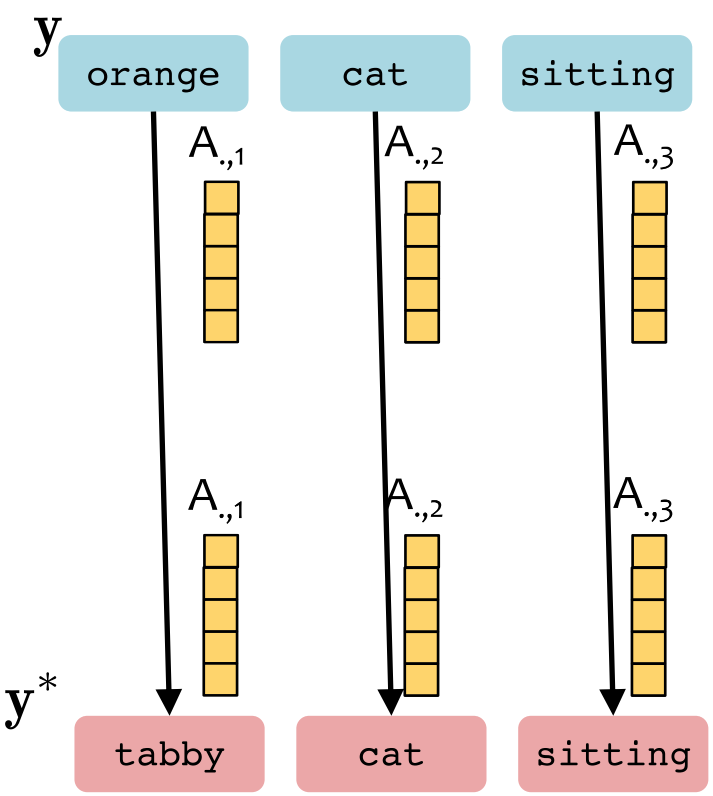
=

A

1	6	11
2	7	12
3	8	13
4	9	14
5	10	15

M

1	0	0
0	1	0
0	0	1



Attention Swapping

- We need to do this swapping for every batch, for every head, and for every timestep (until tau)
- Each row corresponds to a different latent space dimension
- Efficiency trick: define a mapper matrix M such that Â = AM
- Instead of copying, we average over "big" and "tabby"

Â

1/2	1/2	6	11
2/2	2/2	7	12
3/2	3/2	8	13
4/2	4/2	9	14
5/2	5/2	10	15

=

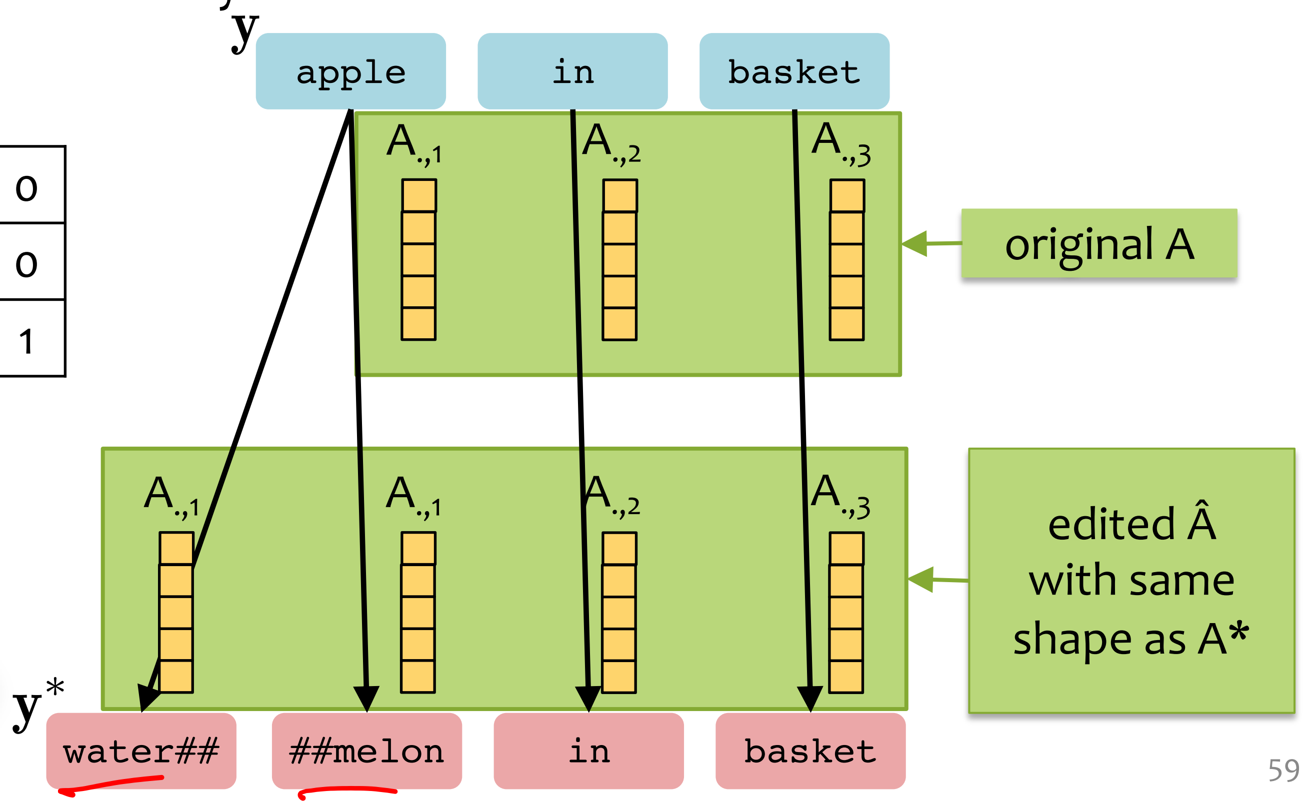
A

1	6	11
2	7	12
3	8	13
4	9	14
5	10	15

M

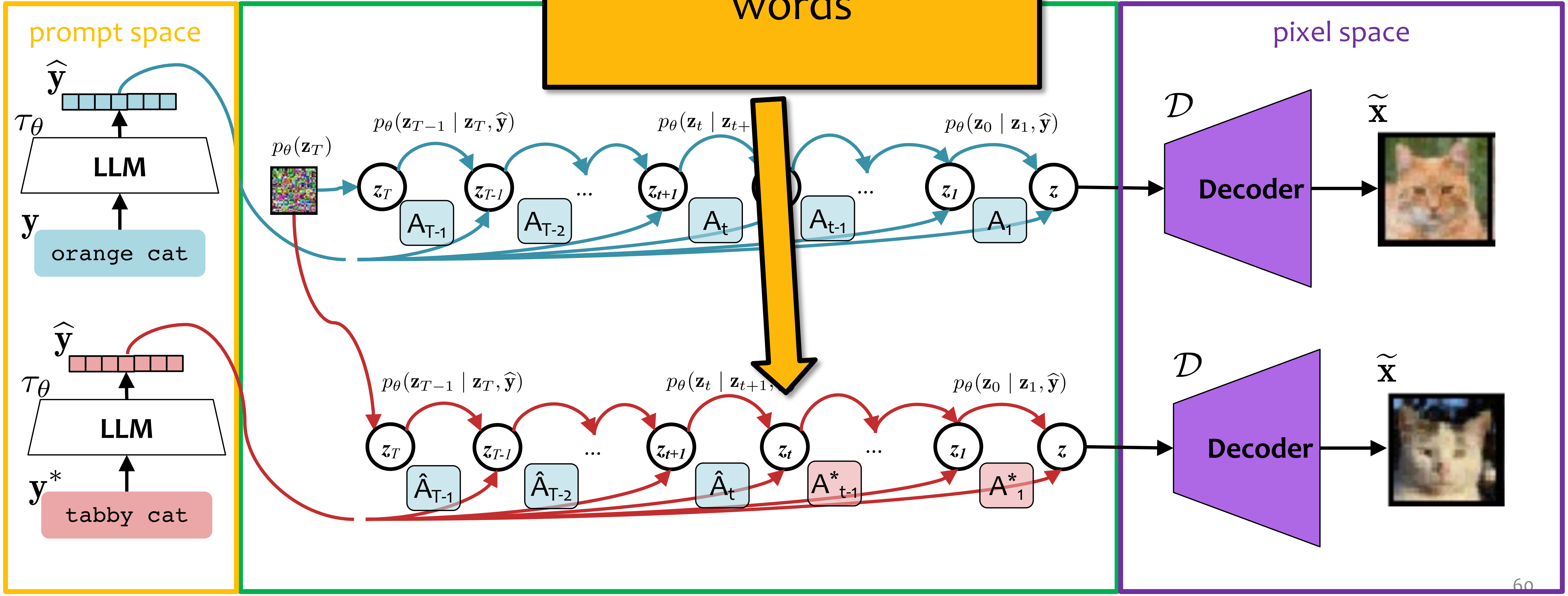
.5	.5	0	0
0	0	1	0
0	0	0	1

The same approach applies if you assume the same number of words since one word might consist of multiple tokens



Prompt-to-Prompt: Alpha

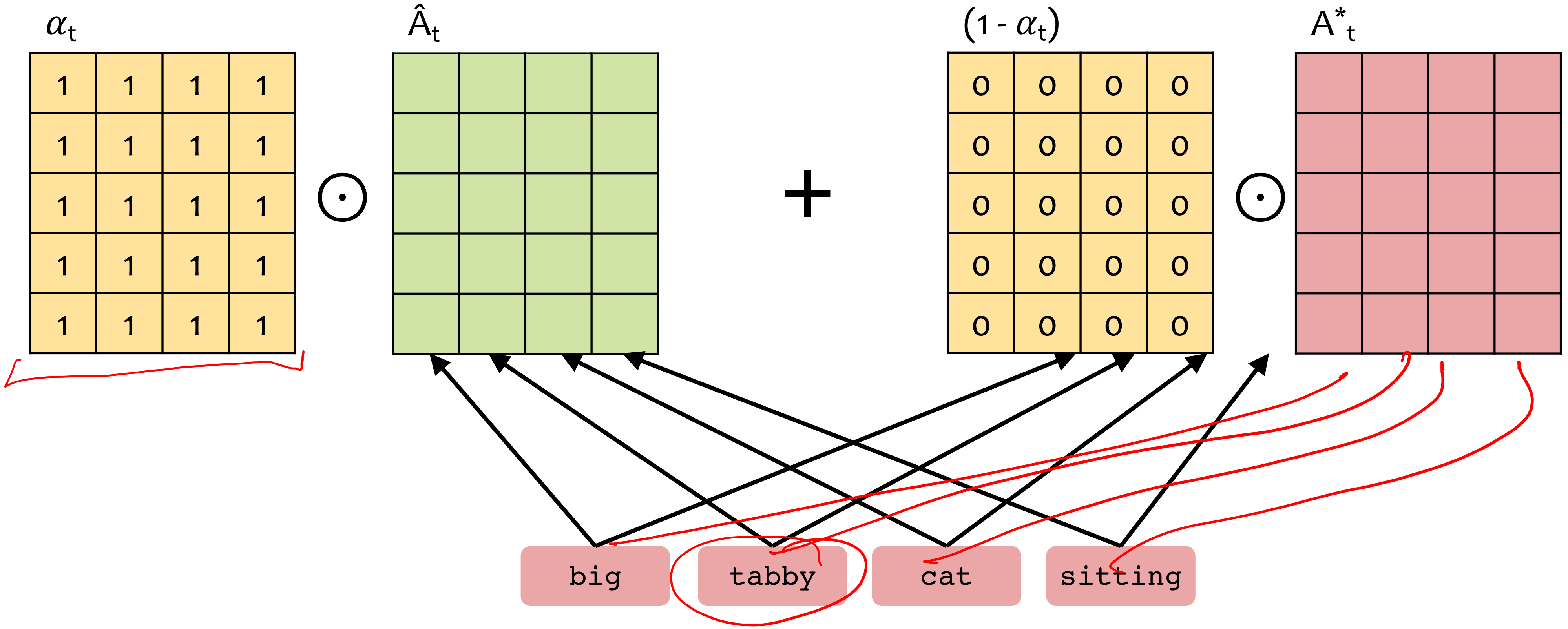
Hyperparameter tuning: we can swap from Â to A* at different timesteps for different words



Prompt-to-Prompt: Alpha

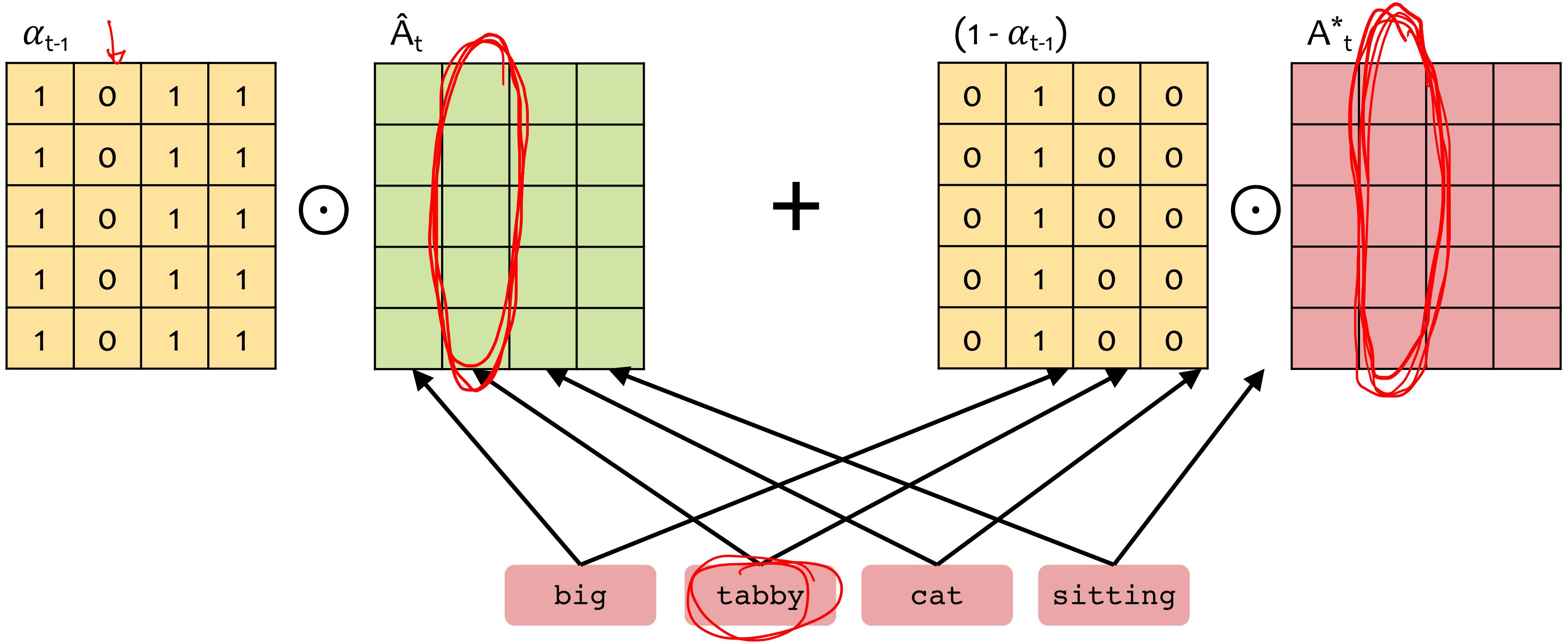
- We can swap from Â to A* at different timesteps for different words
- New hyperparameters: $\alpha_T, \ldots, \alpha_1$
- The matrix α_t controls how/when we switch at timestep t
- For example, if we want to allow one word to deviate from the attention pattern earlier than the others, then that word's column can change before the others

$$\alpha_t \odot \hat{A}_t + (1-\alpha_t) \odot A^*_t$$



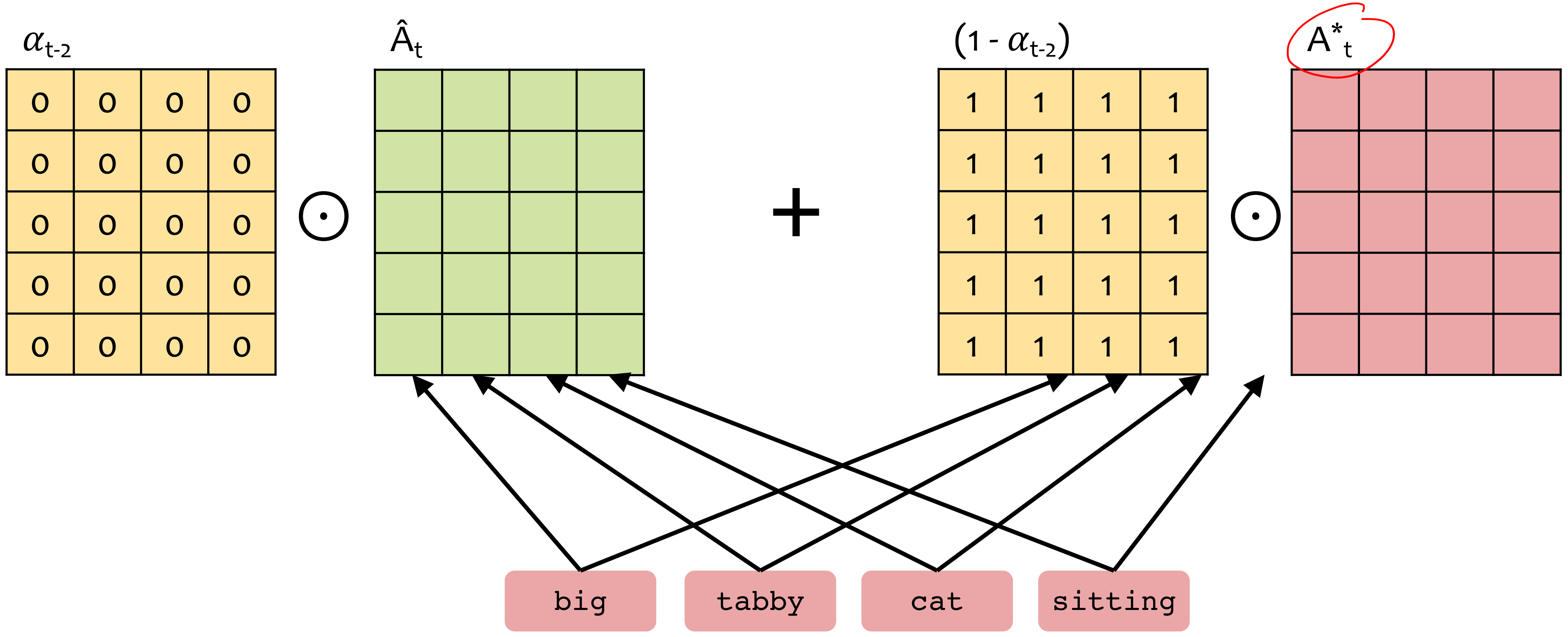
Prompt-to-Prompt: Alpha

- We can swap from Â to A* at different timesteps for different words
- New hyperparameters: $\alpha_T, \ldots, \alpha_1$
- The matrix α_t controls how/when we switch at timestep t
- For example, if we want to allow one word to deviate from the attention pattern earlier than the others, then that word's column can change before the others



Prompt-to-Prompt: Alpha

- We can swap from Â to A* at different timesteps for different words
- New hyperparameters: $\alpha_T, \ldots, \alpha_1$
- The matrix α_t controls how/when we switch at timestep t
- For example, if we want to allow one word to deviate from the attention pattern earlier than the others, then that word's column can change before the others



Prompt-to-Prompt Results

- word/phrase cross-attention swapping automatically identifies the regions of the image that need to remain constant and those that should be adapted

Figure 5: Object preservation. By injecting only the attention weights of the word "butterfly", taken from the top-left image, we can preserve the structure and appearance of a single item while replacing its context. Note how the butterfly sits on top of all objects in a very plausible manner.

Figure from http://arxiv.org/abs/2208.01626

Prompt-to-Prompt Results

- varying the moment of the attention swap to A* allows us to see the effect of our cross-attention manipulation

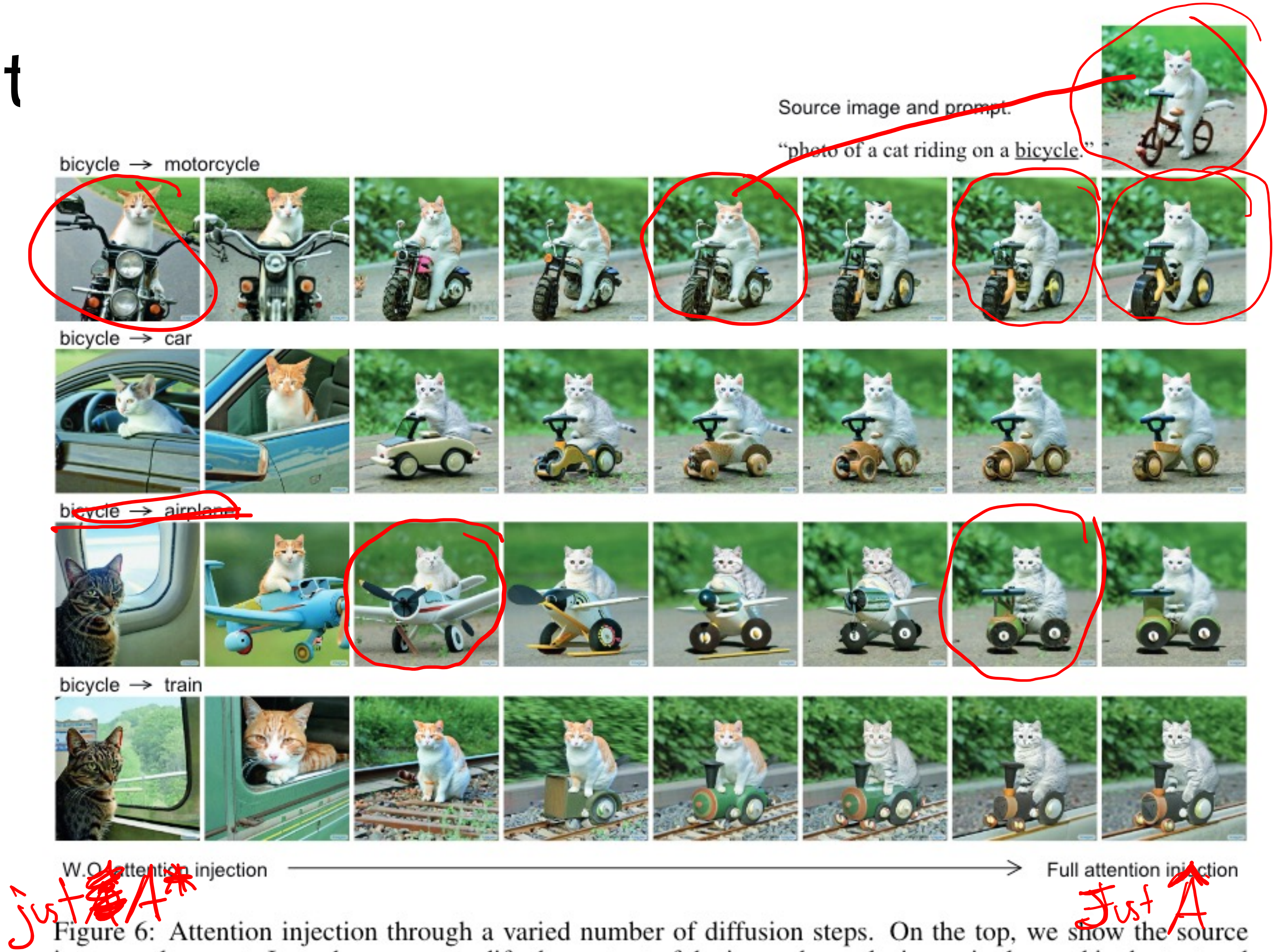


Figure 6: Attention injection through a varied number of diffusion steps. On the top, we show the source image and prompt. In each row, we modify the content of the image by replacing a single word in the text and injecting the cross-attention maps of the source image ranging from 0% (on the left) to 100% (on the right) of the diffusion steps. Notice that on one hand, without our method, none of the source image content is guaranteed to be preserved. On the other hand, injecting the cross-attention throughout all the diffusion steps may over-constrain the geometry, resulting in low fidelity to the text prompt, e.g., the car (3rd row) becomes a bicycle with full cross-attention injection.

Figure from http://arxiv.org/abs/2208.01626

Prompt-to-Prompt Results

- So far we've focused on swapping one word/phrase for another
- Prompt-to-prompt supports different types of edits
- Different types of edits are achieved through different manipulations of cross-attention weights

Figure from http://arxiv.org/abs/2208.01626