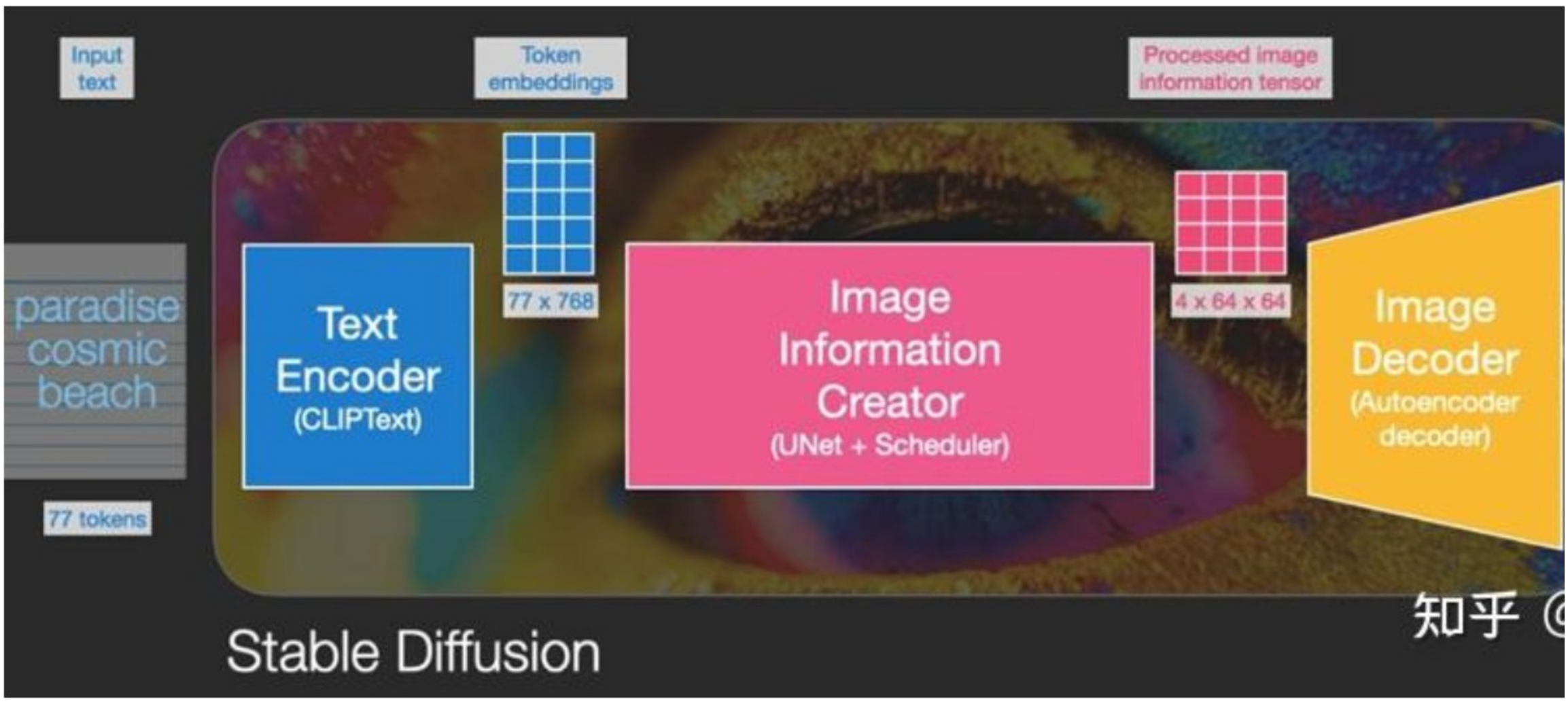
stable diffusion模型分析

Strong
低学历高脂肪算法工程师

AIGC领域大火的stable diffusion模型由文本编码器，图像信息生成器，图像解码器三
（JayAlammar博客）。



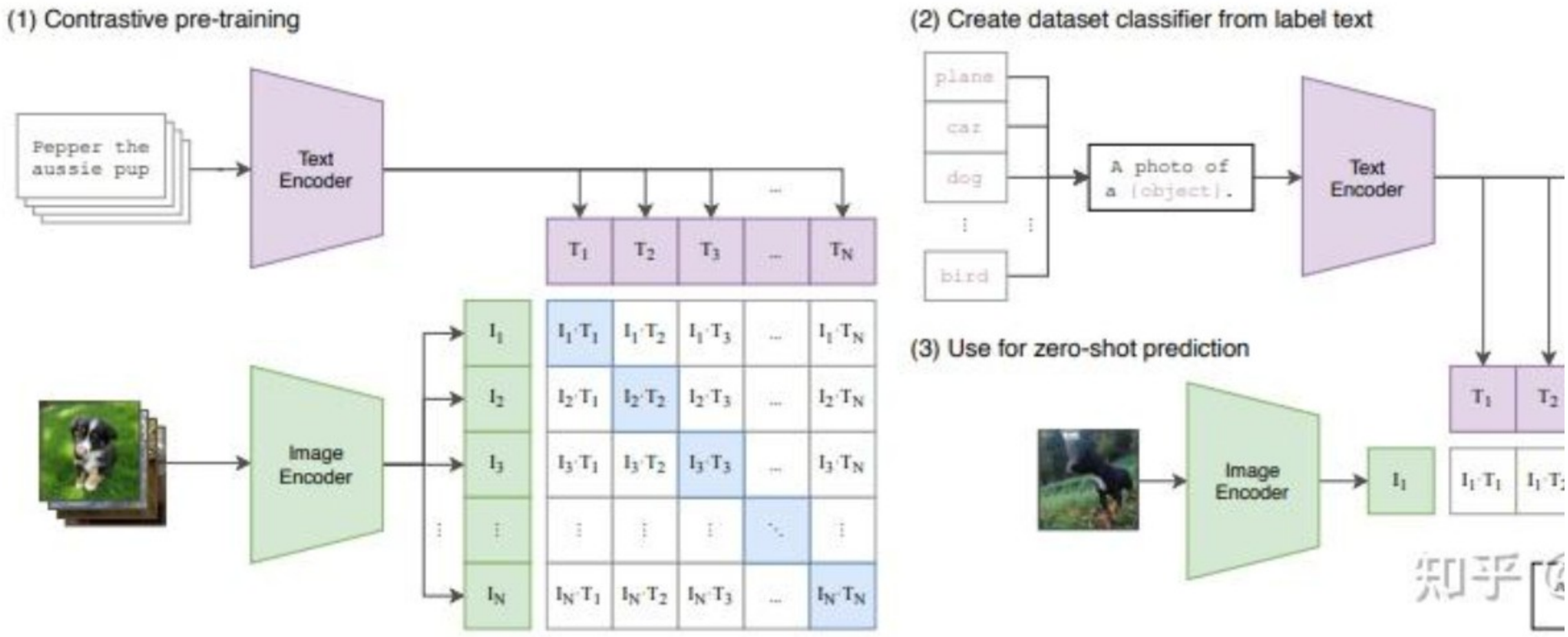
JayAlammar博客中介绍的stable diffusion流程

首先对这三部分模型分别进行分析，然后介绍一下inference sample的代码流程和用te
加速的一些研究。

文本模型

文本编码器是一种Transformer语言模型，作为语言理解组件，接收文本提示，生成词
的Stable Diffusion模型使用ClipText（基于GPT的模型），而论文中使用BERT。

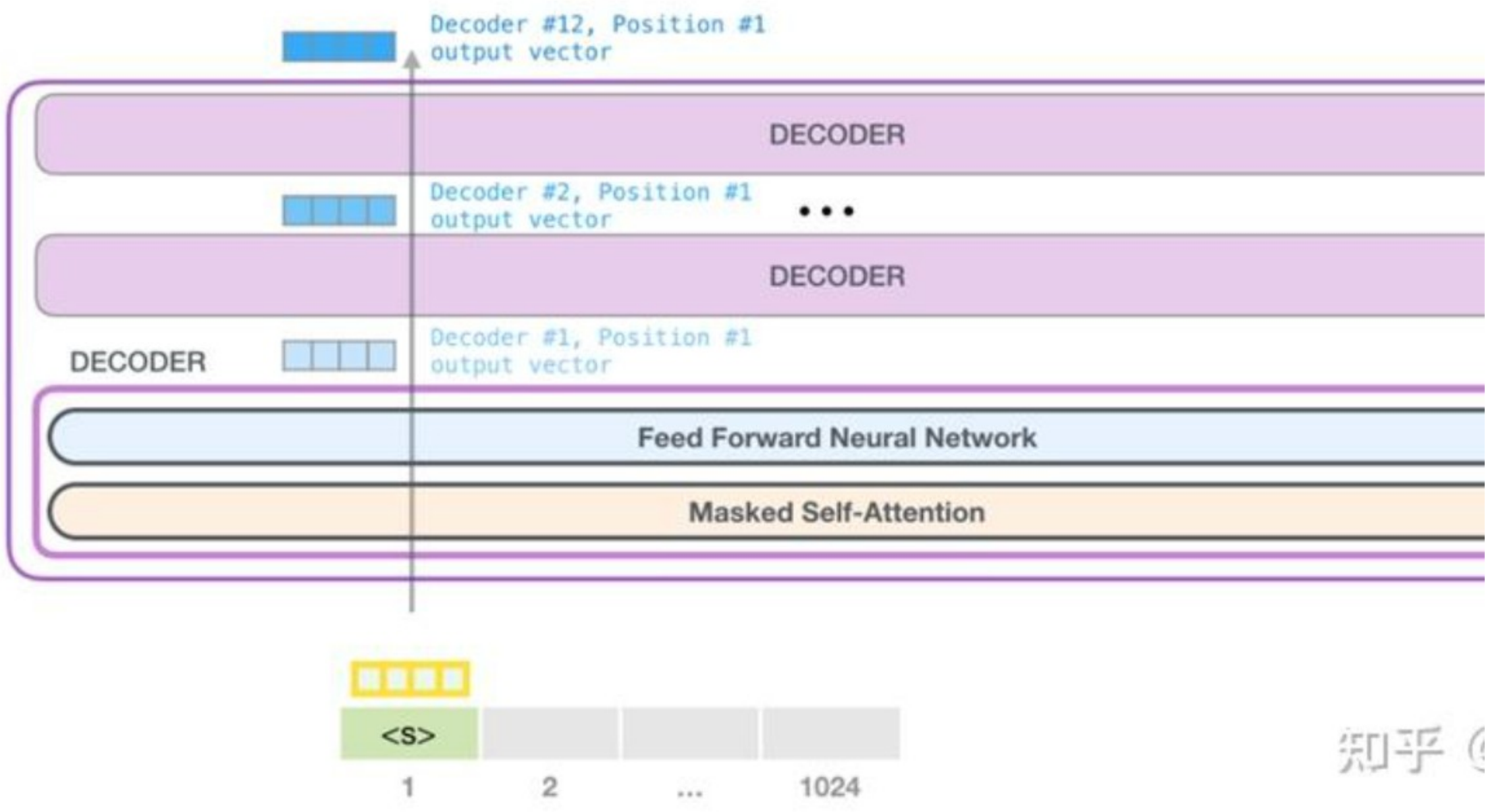
要了解Text Encoder，首先简单介绍一下CLIP这个模型。该模型的训练集由 4 亿张图
组成，训练时分别对图像和文本进行编码，然后，使用余弦相似度比较生成的嵌入。
使编码器能够生成图像和描述相似的嵌入。总结起来如下图所示：



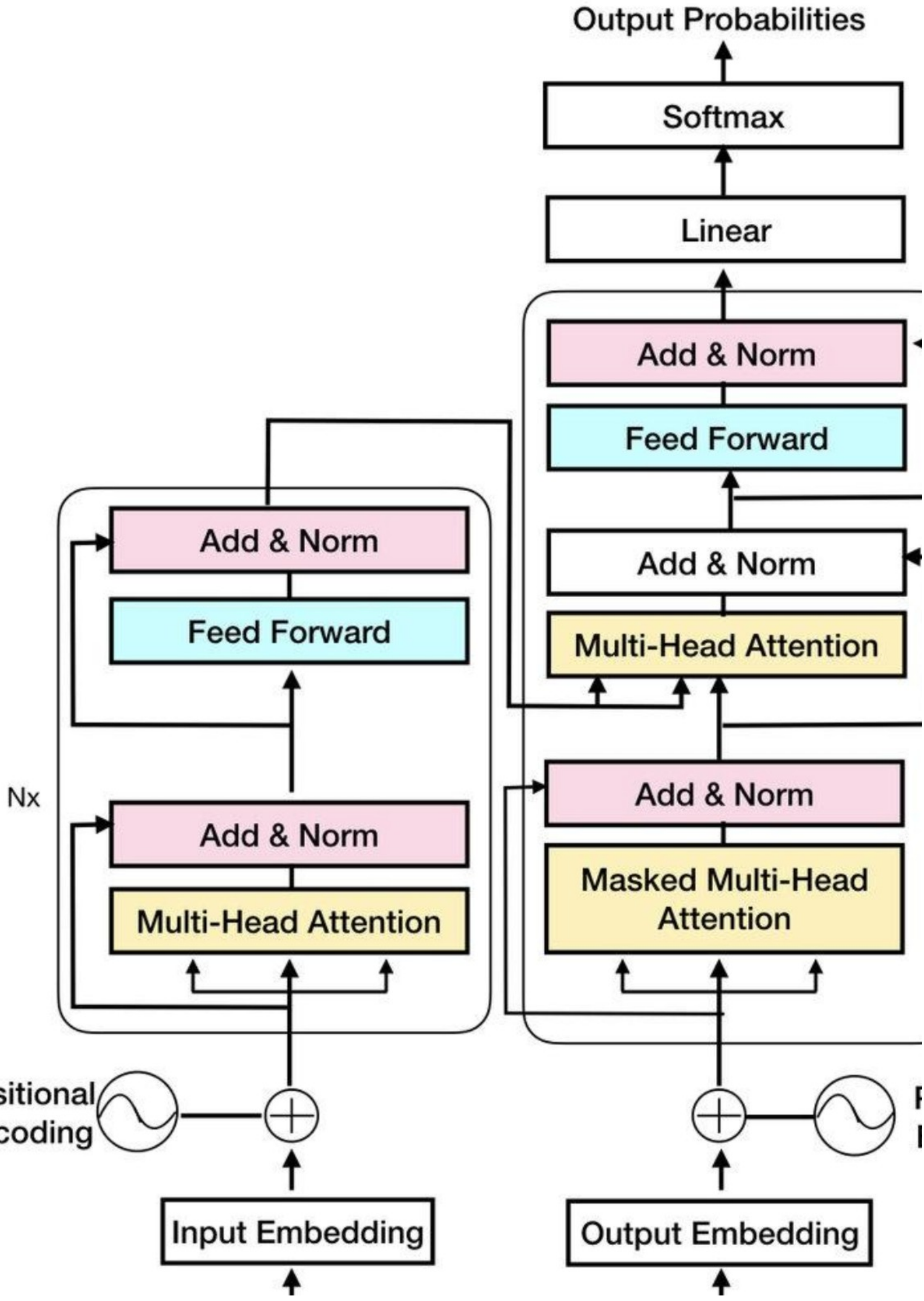
CLIP模型流程图

可以看出图中标蓝色的矩阵对角线都是正样本，其他的就是负样本。实际推理预测的时像来了，经过image encoder得到图片嵌入向量，将该向量和所有label文本生成的嵌入softmax就可以得到该图片分属于哪一个label文本，从而完成该图片的分类。整体过程单清晰的。

stable diffusion中使用的就是clip的text encoder部分，原始的clip是采用GPT-2模型来务。注意，GPT-2模型由transformer的decoder模块堆叠而成，但是和标准transform又有些不同之处：



GPT-2结构



标准transformer结构

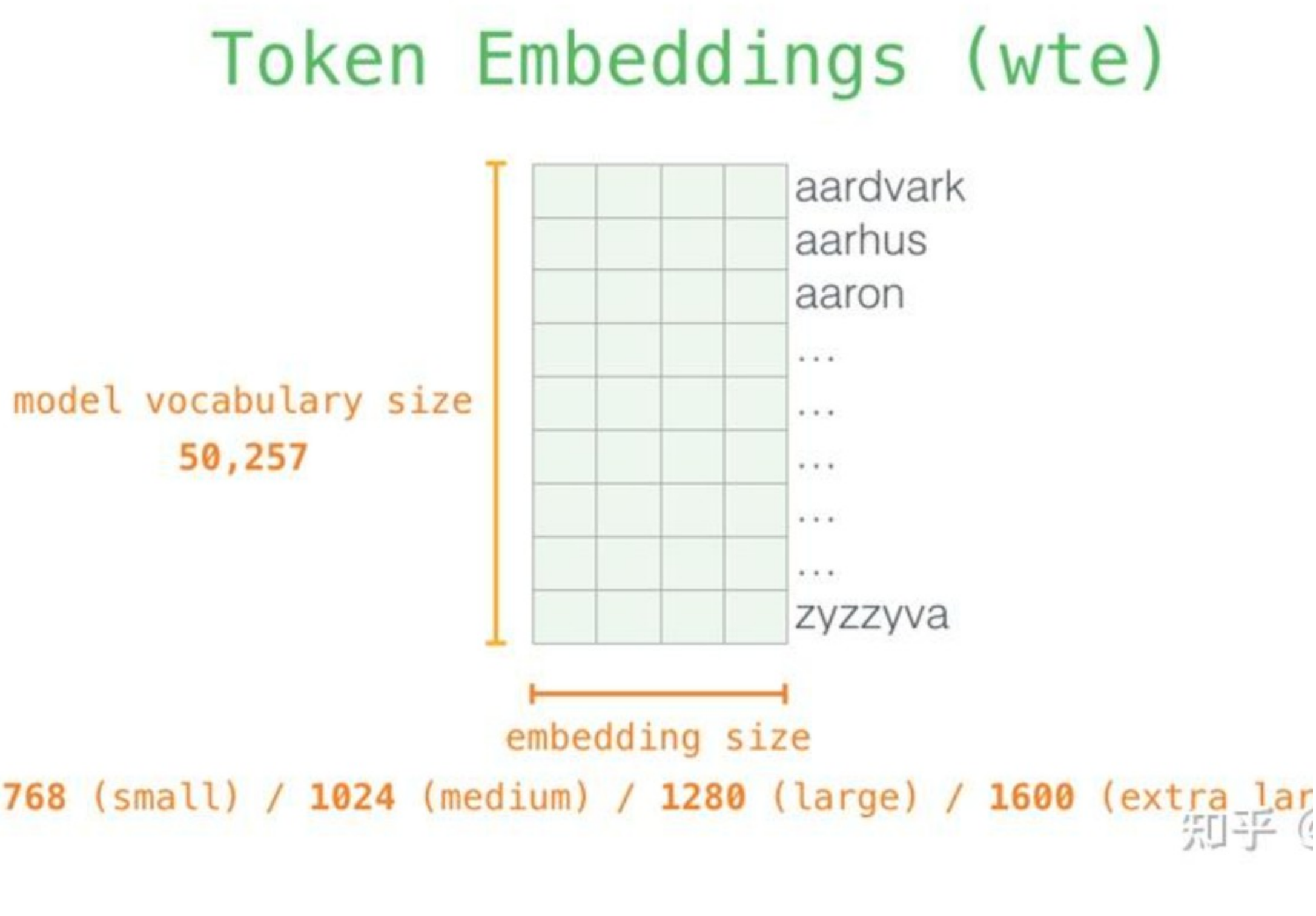
可以看到，GPT中的decoder和标准decoder相比，是没有第二个multi-head attentior可以叫做encode-decode cross attention）。

总结起来，GPT-2是这样一个过程，首先通过该单词的id去token embeddings字典中去的嵌入向量，接着通过该单词的position去positional字典中去查询对应的嵌入向量，然量相加，送入第一层decoder，接着第二层，第三层。。。一般最小的GPT-2有12层de完第12层，还是得到一个嵌入向量，用它与token embeddings字典相乘，得到长度为一个序列，接着进行softmax取到一个最大分数的单词。接着进行下一个单词的处理。

至于masked self-attention的概念，是说在每个单词计算qxk的时候，需要和其他单词数，它只和它前面的单词产生关系，后面的单词都被掩盖住了。比如当前处理的是第1个句子长度为20），那么它的q是1x768这样一个向量，它会和它自己的k和前面9个单（k也是1x768的向量），得到10个分数，而后面10个单词它是不去处理的。然后再和的v向量相乘，得到10个v，再累加得到z。注意，前面单词的k和v向量，并不是处理第时候计算出来的，而是在前面的单词处理过程中缓存住的。

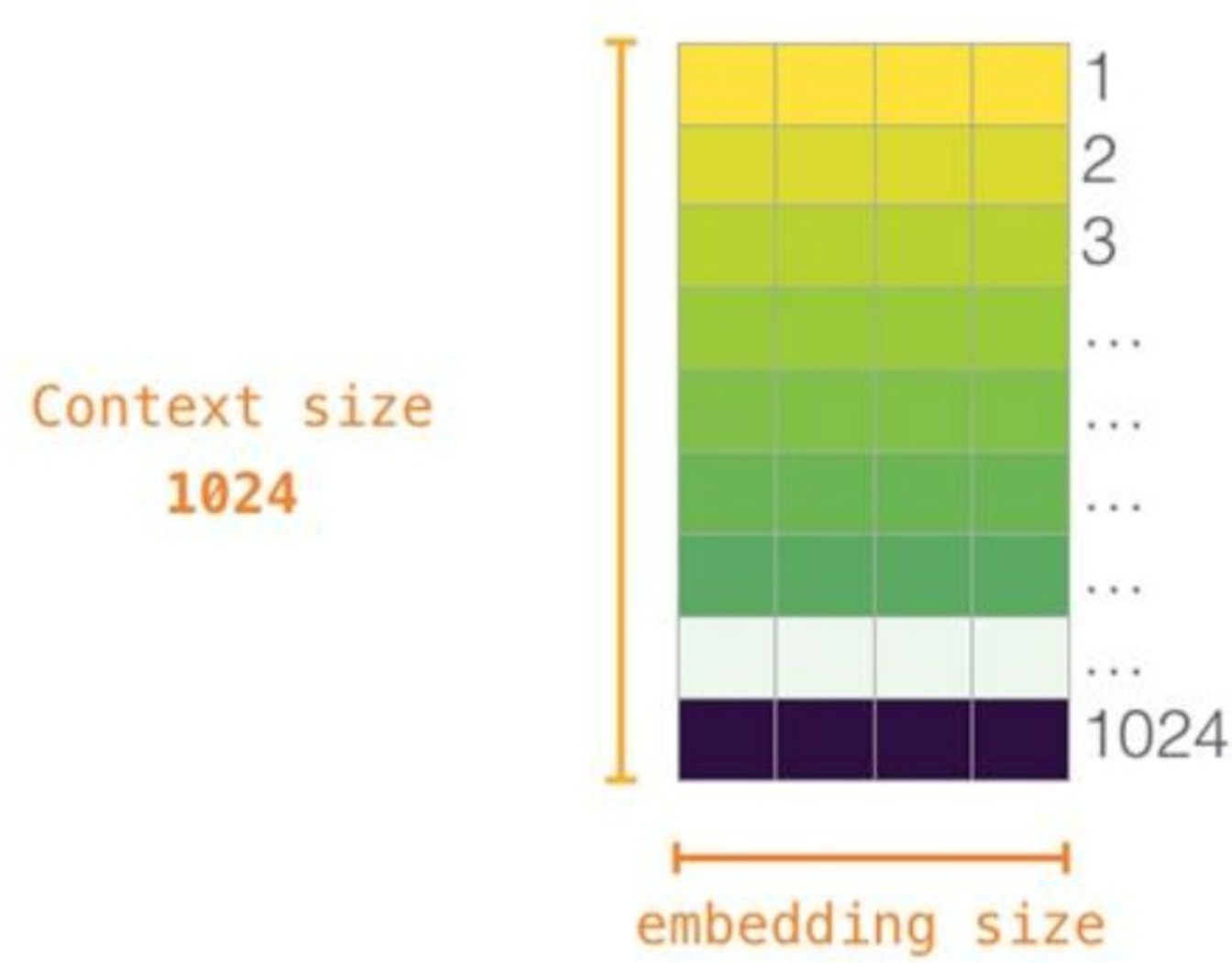
第一个单词的输入

其中，token embeddings一般是一个比较大的词典：



长度为50257的词典

Position embedding是一个长度为1024的词典：



长度为1024的词典

cliptextmodel参数量计算（以hugging face的diffusers工程为例，长度为77，hidden-12层decoder）：

token_embedding： 49408x768

position_embedding: 77x768

每层k weight：768x768 bias：1x768

每层q weight：768x768 bias：1x768

每层v weight：768x768 bias：1x768

每层 out weight：768x768 bias：1x768

每层layer_norm0 weight: 1x768 bias：1x768

每层 ffn0 weight：3072x768 bias：1x3072

每层 ffn1 weight：768x3072 bias：1x768

每层layer_norm1 weight: 1x768 bias：1x768

汇总：

```
Total =
49408x768+
77x768+
12x(4x768x768+4x768+2x768+2x3072x768+3072+768+2x768)
=123059712
```

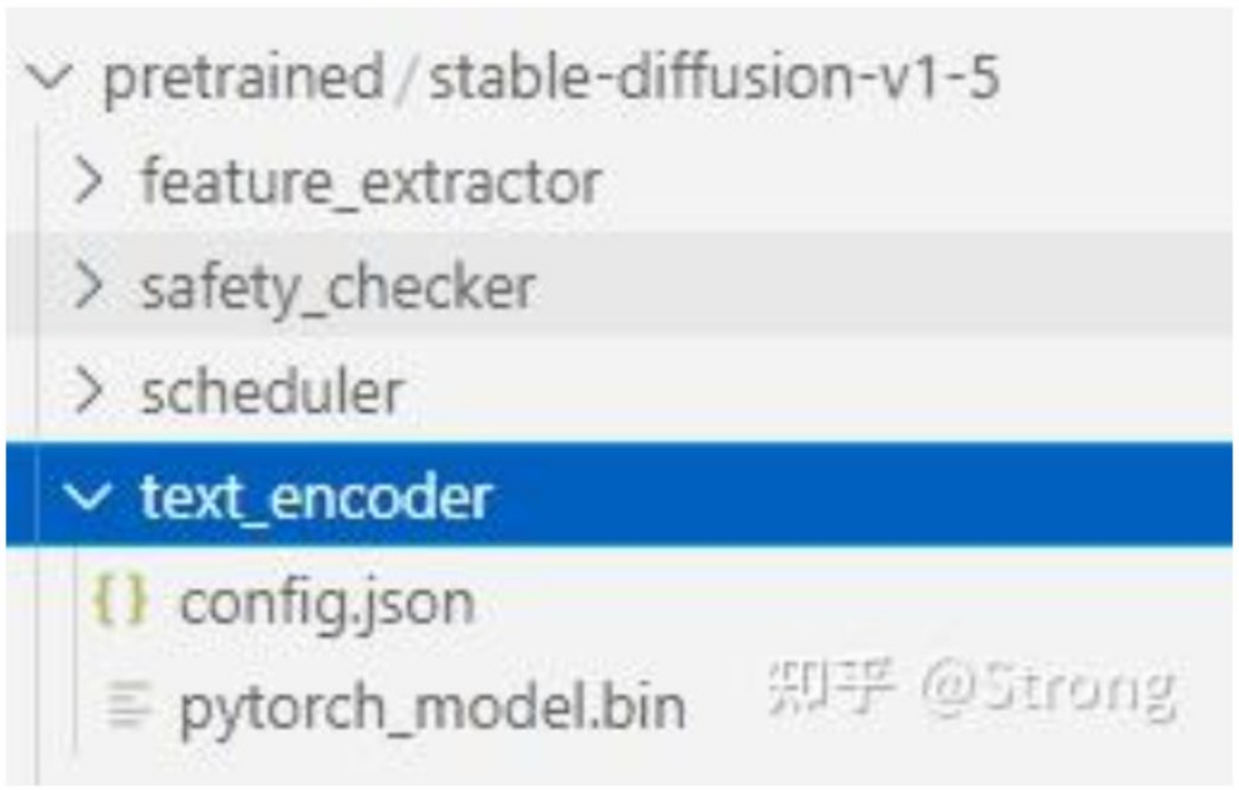
以float32来计算，参数量大概是 123059712x4/1024/1024 = 469M Byte。其中，字典
144M，decoder参数占了324M。至于clip论文中说用的text模型是63M个参数，我觉
为它用的hidden size为512导致。

模型可以按照diffuers工程的guide下载

下载模型之前需先安装git lfs

```
curl -s https://packagecloud.io/install/repositories/github/git-lfs/
sudo apt-get install git-lfs
git lfs install
下载模型：  git clone https://huggingface.co/runwayml/stable-diffusion-
```

下载的模型中有每个部分的模型ckpt，.bin文件可以用netron工具进行可视化



下载的stable-diffusion模型

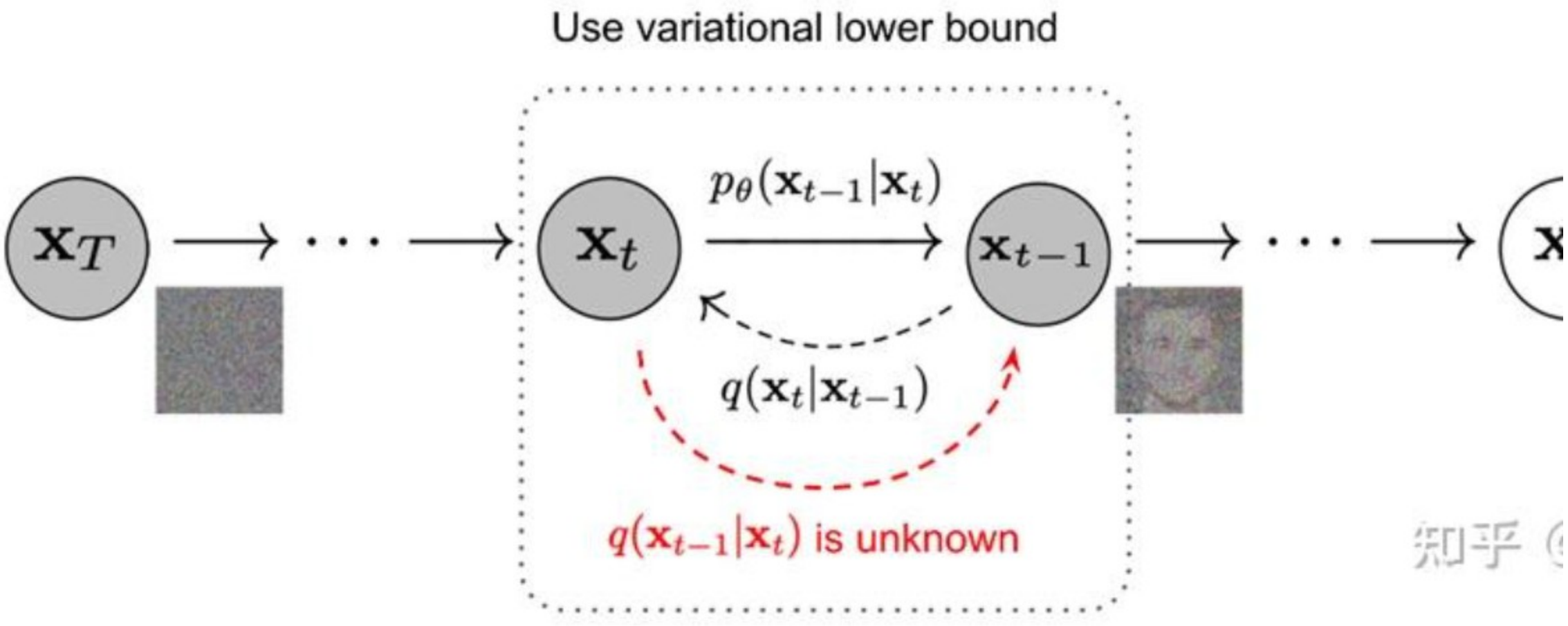
Image Creator

前向过程

Creator代表扩散模型的反向过程。这里，我们先了解以下扩散模型的前向过程。所谓即往图片上加噪声的过程。考虑向一个从真实数据分布中随机采样的变量添加噪声，添后，会得到一个长度为T的序列，随着T的增大，原始数据会丢失它的特征而变成一个纯声。

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \quad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}$$

在这个过程中，每个时刻t只与t-1时刻有关，所以可以看作一个马尔可夫过程。



虽然xt只与xt-1相关，但是通过独立高斯分布性质和重参数化技巧，经过推导，可以得于x0的关系：

$$\text{Let } \alpha_t = 1 - \beta_t \text{ and } \bar{\alpha}_t = \prod_{i=}^{t}$$

$$\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1} && \text{;where } \boldsymbol{\epsilon}_{t-1}, \boldsymbol{\epsilon}_{t-2}, \cdots \\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\bar{\boldsymbol{\epsilon}}_{t-2} && \text{;where } \bar{\boldsymbol{\epsilon}}_{t-2} \text{ merges two Gau} \\
&= \dots \\
&= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon} \\
q(\mathbf{x}_t|\mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})
\end{aligned}$$

$$\boldsymbol{x}_t = \sqrt{\bar{a}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{a}_t}\boldsymbol{z}$$

反向过程

反向过程就是去噪的过程。知道xt来计算xt-1。但是我们对于q(xt-1|xt)是不知道的，最过推导，得到一个xt-1关于xt的关系。通过贝叶斯法则，可以推导：

$$\begin{aligned}
q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) &= q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)\frac{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} \\
&\propto \exp\Big(-\frac{1}{2}\big(\frac{(\mathbf{x}_t - \sqrt{\alpha_t}\mathbf{x}_{t-1})^2}{\beta_t} + \frac{(\mathbf{x}_{t-1} - \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0)^2}{1-\bar{\alpha}_{t-1}} - \frac{(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0)^2}{1-\bar{\alpha}_t}\big)\Big) \\
&= \exp\Big(-\frac{1}{2}\big(\frac{\mathbf{x}_t^2 - 2\sqrt{\alpha_t}\mathbf{x}_t\mathbf{x}_{t-1} + \alpha_t\mathbf{x}_{t-1}^2}{\beta_t} + \frac{\mathbf{x}_{t-1}^2 - 2\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0\mathbf{x}_{t-1} + \bar{\alpha}_{t-1}\mathbf{x}_0^2}{1-\bar{\alpha}_{t-1}} - \frac{(\mathbf{x}_t} \\
&= \exp\Big(-\frac{1}{2}\big((\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_{t-1}})\mathbf{x}_{t-1}^2 - (\frac{2\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t + \frac{2\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}\mathbf{x}_0)\mathbf{x}_{t-1} + C(\mathbf{x}_t, \mathbf{x}_0)\big)\Big)
\end{aligned}$$

可以看到，经过推导变成了我们正向过程中已经可以表达的关系。但是反向过程中我们不知道的。但根据正向中xt与x0的关系，我们可以把x0进行替换，最终得到只与xt和-

的关系：

$$\tilde{\boldsymbol{\mu}}_t = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1-$$

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

至于这个未知的ε，使用深度学习模型（unet，或者最新的DiT论文中用transformer模
unet）来进行预测。

总结起来，creator的每一步推断就是执行下面的算法流程：

1）使用深度学习模型（unet），通过xt和t来预测高斯噪声zθ(xt,t),然后根据公式计算
差：

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{a_t}}(x_t - \frac{\beta_t}{\sqrt{1-\overline{a}_t}}z_\theta(x$$

2） 根据公式得到p(xt-1|xt):

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x$$

3）利用重参数化得到xt-1，重参数化可表示为可求导的分布随机采样。

$$z = \mu_\theta + \sigma_\theta \odot \epsilon, \epsilon \sim \mathcal{N}(0$$

其中⊙表示element-wise乘。

代码流程

对应到hugging face工程中的代码，上述计算xt-1的过程就是不同的scheduler，stable pipeline中支持三种scheduler，分别为：

PNDM scheduler (used by default)

DDIM scheduler

K-LMS scheduler

以DDIM为例，计算xt-1的公式和上述介绍有些不同，但是大体都是类似的：

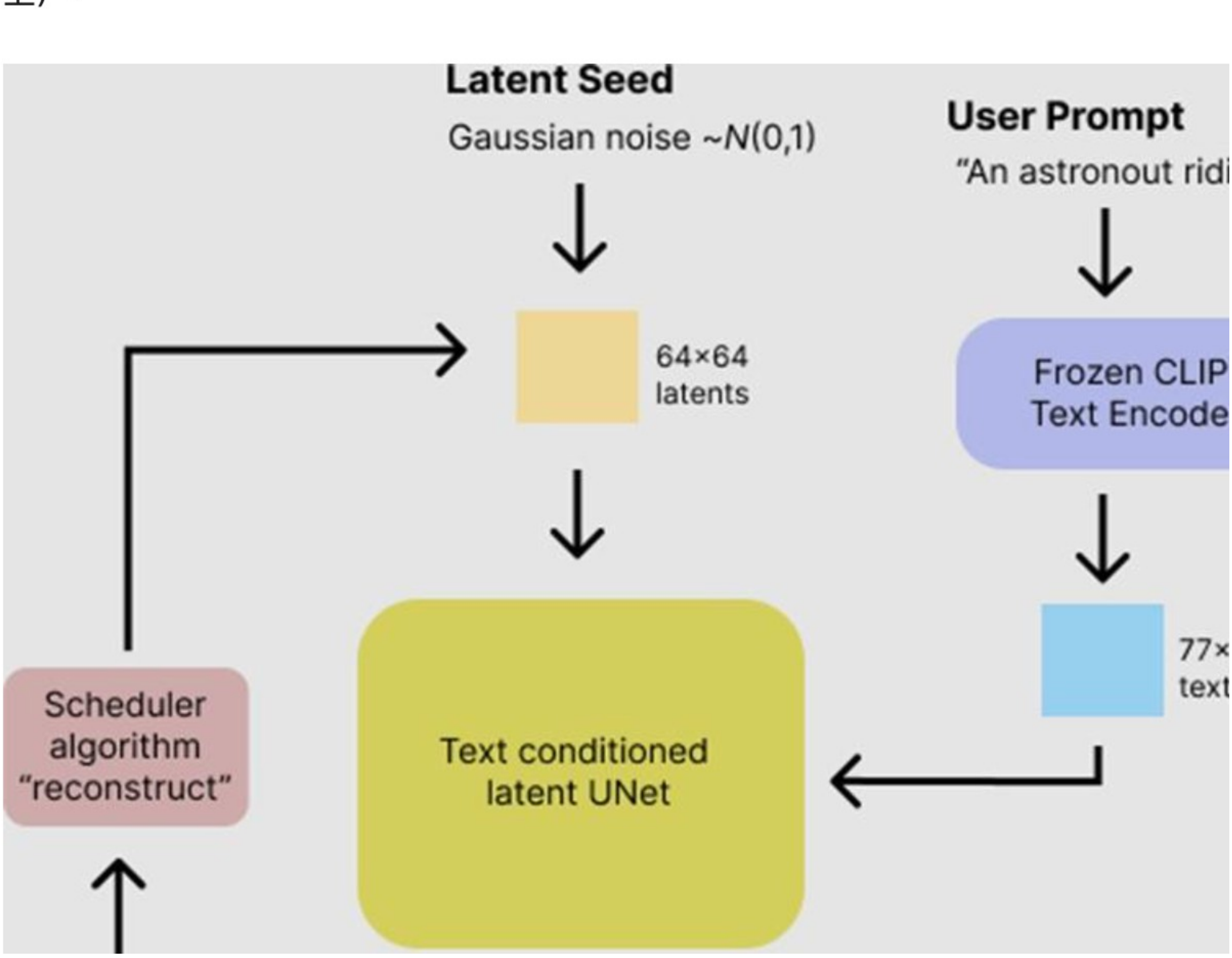
$$\boldsymbol{x}_{t-1} = \sqrt{\alpha_{t-1}} \underbrace{\left(\frac{\boldsymbol{x}_t - \sqrt{1-\alpha_t}\epsilon_\theta^{(t)}(\boldsymbol{x}_t)}{\sqrt{\alpha_t}} \right)}_{\text{“ predicted } \boldsymbol{x}_0\text{”}} + \underbrace{\sqrt{1-\alpha_{t-1}-\sigma_t^2} \cdot \epsilon_\theta^{(t)}(\boldsymbol{x}_t)}_{\text{“direction pointing to } \boldsymbol{x}_t\text{”}} +$$

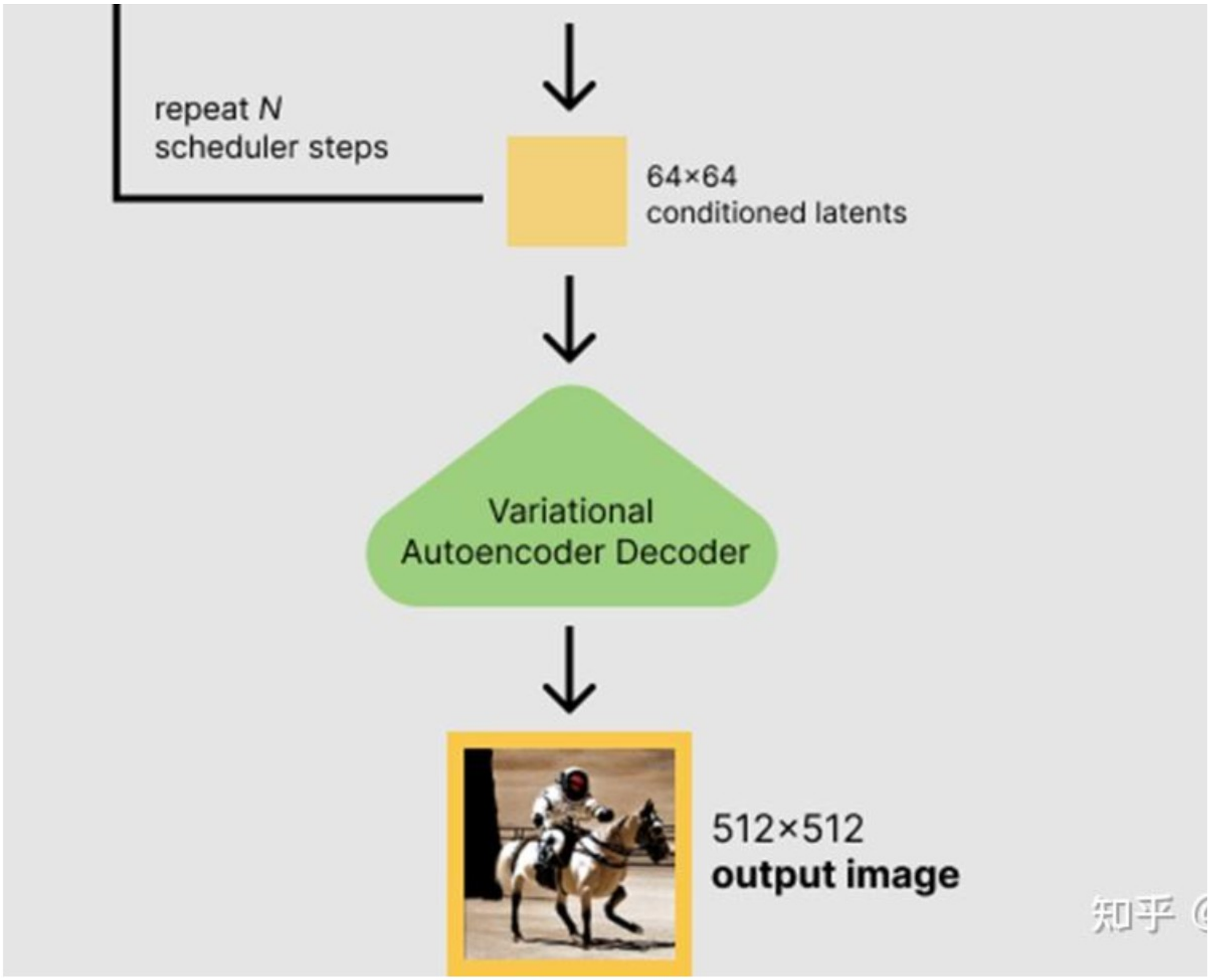
在代码中，其中的εθ(t)就是unet的输出，xt就是latent（即当前iter的整体输入）。截取中step的一小段：

```python
# 3. compute predicted original sample from predicted noise also called
# "predicted x_0" of formula (12) from https://arxiv.org/pdf/2010.02502.pdf
if self.config.prediction_type == "epsilon":
    pred_original_sample = (sample - beta_prod_t ** (0.5) * model_output) / alpha_prod_t ** (
elif self.config.prediction_type == "sample":
    pred_original_sample = model_output
elif self.config.prediction_type == "v_prediction":
    pred_original_sample = (alpha_prod_t**0.5) * sample - (beta_prod_t**0.5) * model_output
    # predict V
    model_output = (alpha_prod_t**0.5) * model_output + (beta_prod_t**0.5) * sample
else:
    raise ValueError(
        f"prediction_type given as {self.config.prediction_type} must be one of `epsilon`, `s
        " `v_prediction`"
    )
```

可以看到就是在计算公式的第一部分x0。

Hugging face中代码整体流程可以用这个图来表示（注意scheduler的过程，其实就是后都要执行一次，得到latent。从上述理论推导可以看出，scheduler才是扩散模型的核用用来在每次计算xt-1时的一个预测噪声计算步骤中需要的，但是计算量的大头都在这上）：

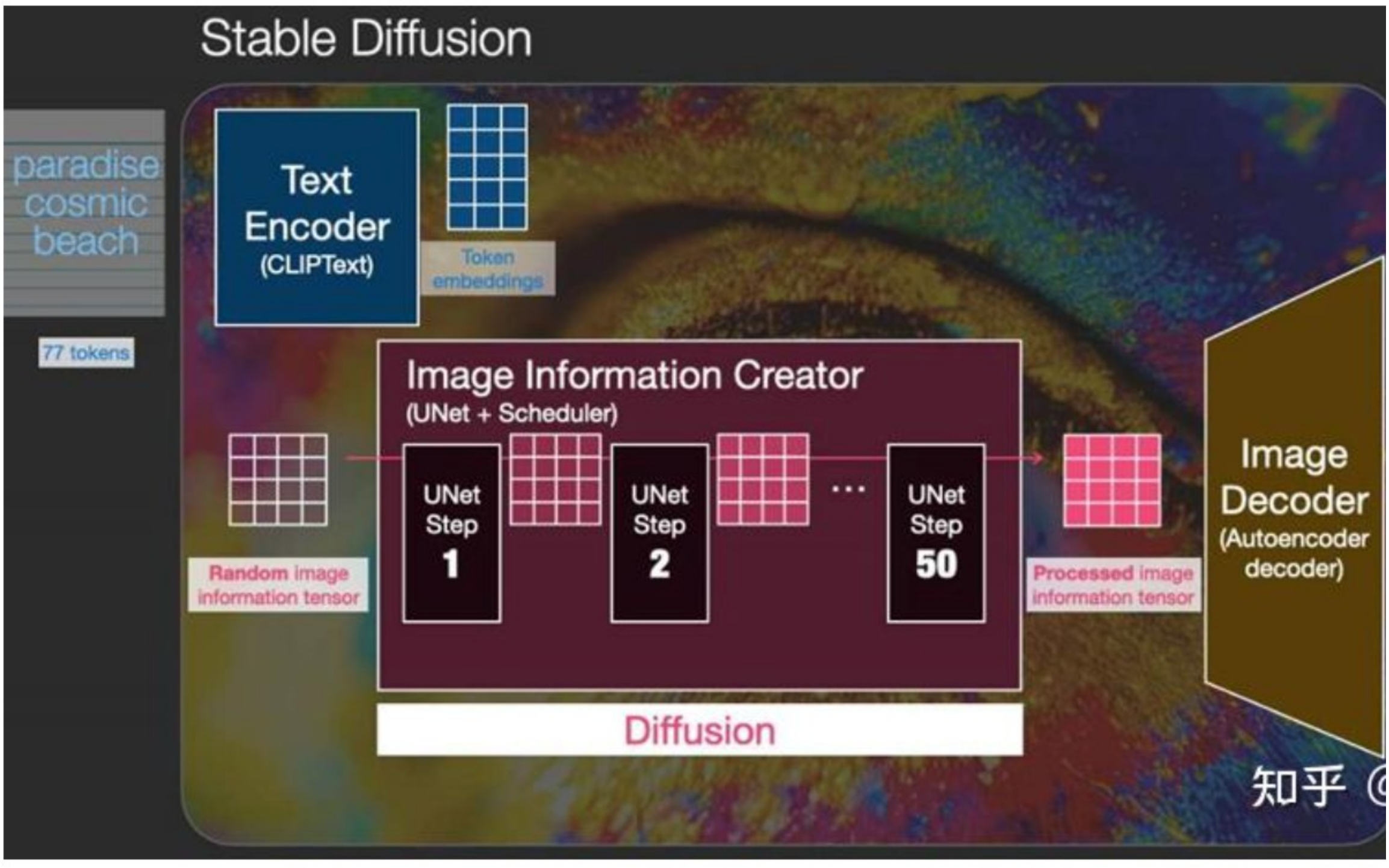




执行流程

公式表达和推导比较繁琐，参考jay的blog，用形象的图文再来介绍一下creator的过程

Creator发生在多个步骤中，每个步骤都对输入的隐信息进行操作，生成另一个隐信息
信息蕴含输入文本信息以及模型从图像训练集获得的视觉信息的融合信息。



可视化一组隐信息状态，查看每个步骤信息变化