Multi Modal LLM

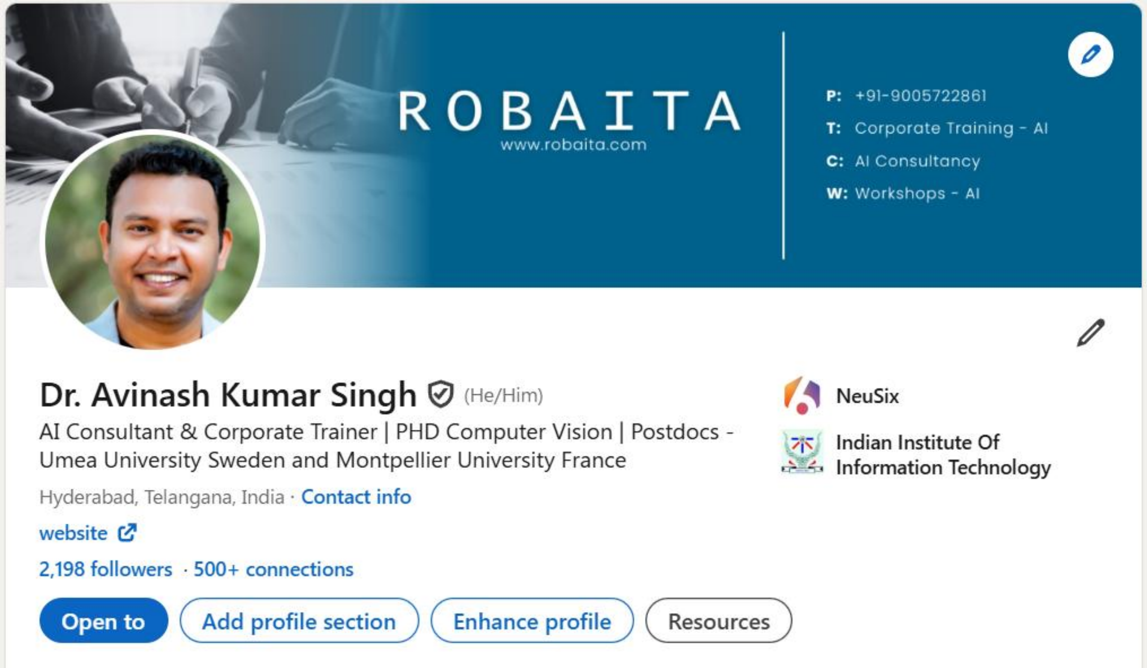
Bridging Text, Vision, and Beyond

Dr. Avinash Kumar Singh

AI Consultant and Coach, Robaita

Dr. Avinash Kumar Singh

- ❑ **Possess** 15+ years of **hands-on expertise** in Machine Learning, Computer Vision, NLP, IoT, Robotics, and Generative AI.
- ❑ **Founded** Robaita—an initiative **empowering** individuals and organizations to **build, educate, and implement** AI solutions.
- ❑ **Earned** a Ph.D. in Human-Robot Interaction from IIIT Allahabad in 2016.
- ❑ **Received** postdoctoral fellowships at Umeå University, Sweden (2020) and Montpellier University, France (2021).
- ❑ **Authored** 30+ research papers in **high-impact** SCI journals and international conferences.
- ❑ Unlearning, learning, making mistakes …



https://www.linkedin.com/in/dr-avinash-kumar-singh-2a570a31/

eClerx Services Limited HCLTech UMEÅ UNIVERSITET BRANE UNIVERSITÉ MONTPELLIER Robaita

Discussion Points

- Vision Transformers
- Multi Model LLM
  - Visual Language Model
    - CLIP
    - BLIP
- Diffusion Models
  - Forward and Reverse Process
  - Latent Diffusion Model
  - Stable Diffusion Model

Vision Transformer (ViT)

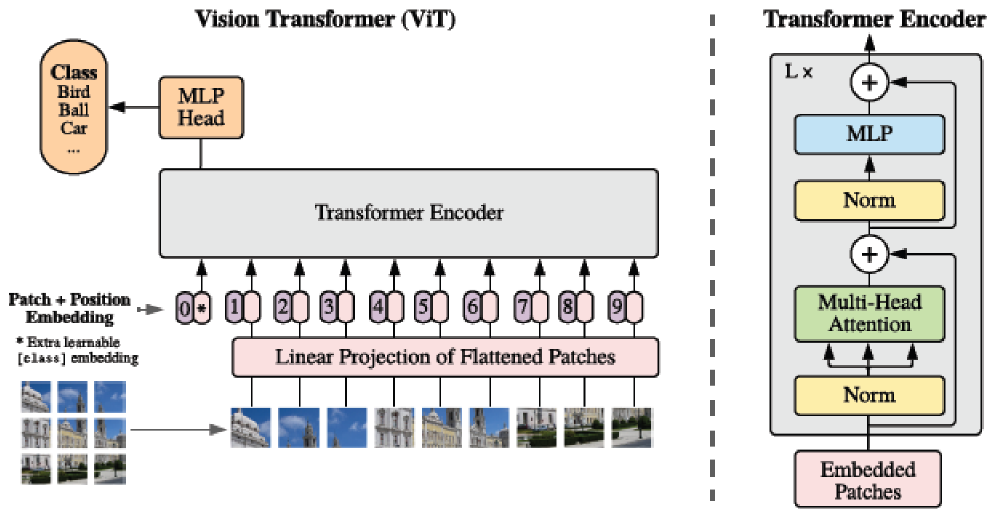


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable “classification token” to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

Published as a conference paper at ICLR 2021

AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

Alexey Dosovitskiy[*,†], Lucas Beyer[*], Alexander Kolesnikov[*], Dirk Weissenborn[*], Xiaohua Zhai[*], Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby[*,†]

[*]equal technical contribution, [†]equal advising

Google Research, Brain Team

`{adosovitskiy, neilhoulsby}@google.com`

ABSTRACT

While the Transformer architecture has become the de-facto standard for natural language processing tasks, its applications to computer vision remain limited. In vision, attention is either applied in conjunction with convolutional networks, or used to replace certain components of convolutional networks while keeping their overall structure in place. We show that this reliance on CNNs is not necessary and a pure transformer applied directly to sequences of image patches can perform very well on image classification tasks. When pre-trained on large amounts of data and transferred to multiple mid-sized or small image recognition benchmarks (ImageNet, CIFAR-100, VTAB, etc.), Vision Transformer (ViT) attains excellent results compared to state-of-the-art convolutional networks while requiring substantially fewer computational resources to train.[1]

Vision Transformer (ViT)

Embedded Patches

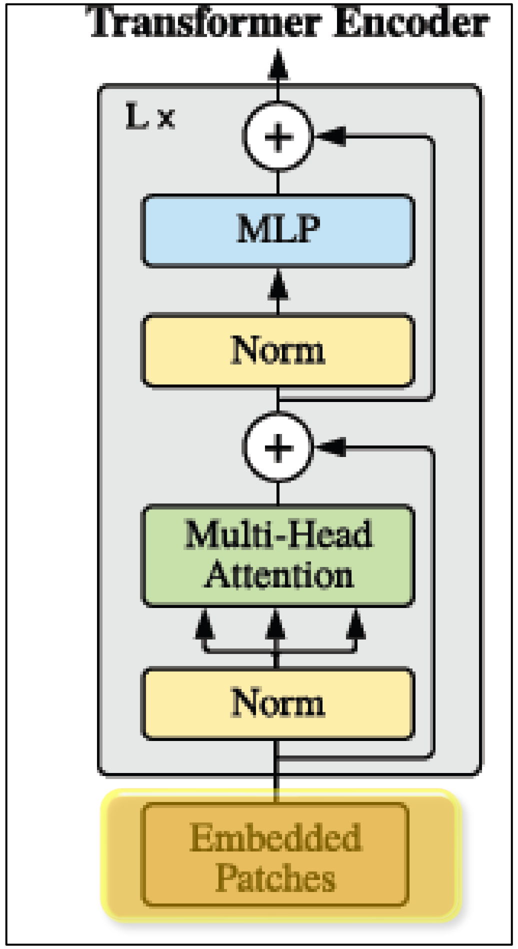
Given an input image $x \in \mathbb{R}^{H \times W \times C}$[H: Height, W:Width, C:Channel], follow the steps

- Divide it into patches of size $P \times P$
- Flatten each patch into a vector. $x \rightarrow \{x_1, x_2, x_3, x_4, \dots, x_N\}$, where $x_i \in \mathbb{R}^{P^2 C}$
- Project each patch into a D-dimensional space using a linear projection.

$$z_i = W_e x_i + b_e, where\ z_i \in \mathbb{R}^D$$

Where $W_e \in \mathbb{R}^{D \times (P^2 * C)}$is the patch embedding matrix, D is the dimension of linear projection matrix

$N = \frac{HW}{P^2}$ is the number of patches



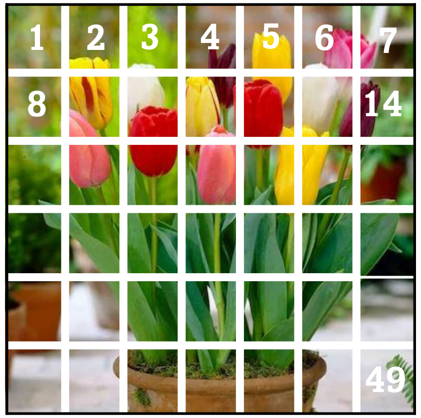
Vision Transformer (ViT)

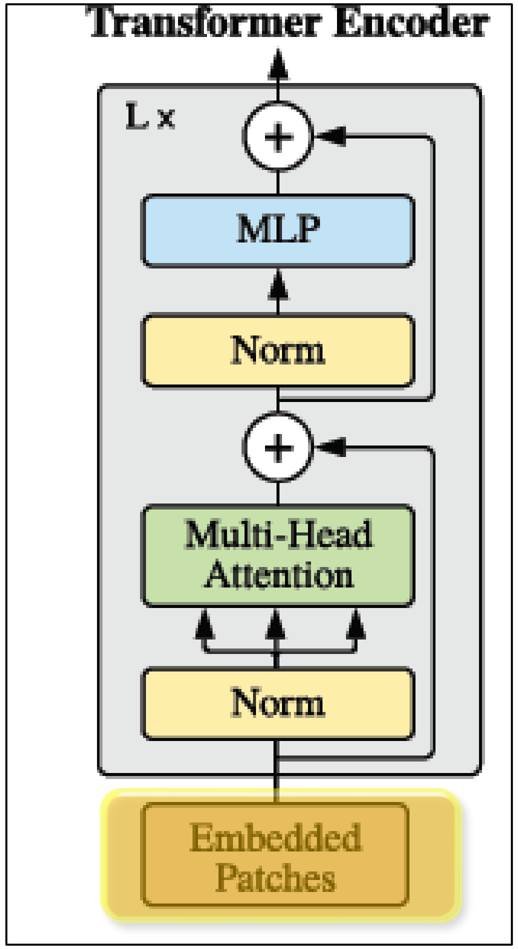
Embedded Patches

Let' say we have an RGB image of 224x224x3.

The embedded patch will be of size

$$B \times H\_P \times W\_P \times emb\_size = 1 \times 14 \times 14 \times 768$$






```python
class PatchEmbedding(nn.Module):
    def __init__(self, in_channels=3, patch_size=16, emb_size=768, img_size=224):
        super().__init__()
        self.patch_size = patch_size
        self.projection = nn.Conv2d(
            in_channels, out_channels=emb_size, kernel_size=patch_size, stride=patch_size
        )

    def forward(self, x):
        x = self.projection(x)  # shape: [B, D, H/P, W/P]
        x = x.flatten(2)        # shape: [B, D, N]
        x = x.transpose(1, 2)   # shape: [B, N, D]
        return x
```

Vision Transformer (ViT)

Positional Encoding

- When you divide an image into patches, each patch is treated like a token.
- However, unlike pixels or sequences in CNNs or RNNs, transformers treat input tokens as orderless.
- To fix this, we add a learnable positional embedding to each patch embedding.

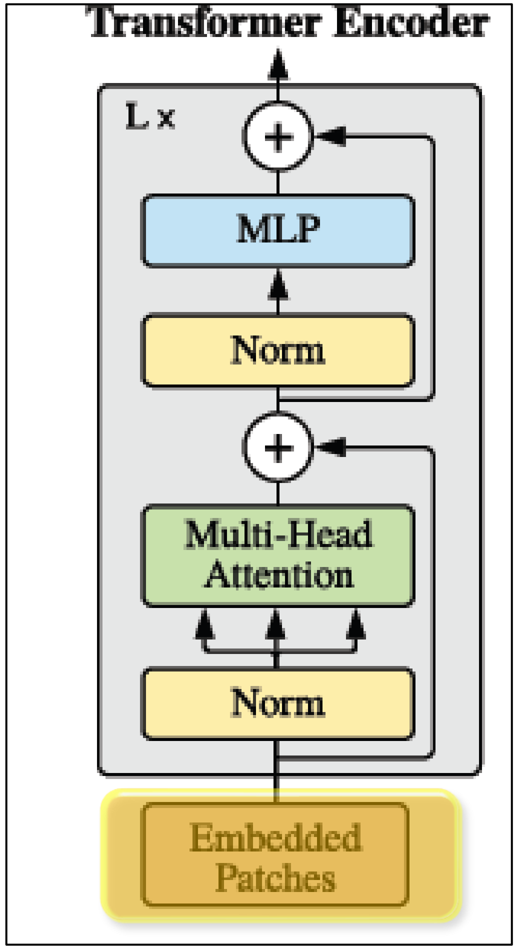
$x_i \in \mathbb{R}^D$ is the encoding of the i^{th}patch

$p_i \in \mathbb{R}^D$ is the learnable positional encoding of the i^{th} location

Final embedding of the i^{th} location

$$z_i = x_i + p_i$$

Where z_i is the input to the transformer for the i^{th} patch (with position awareness).



Vision Transformer (ViT)

Layer Norm

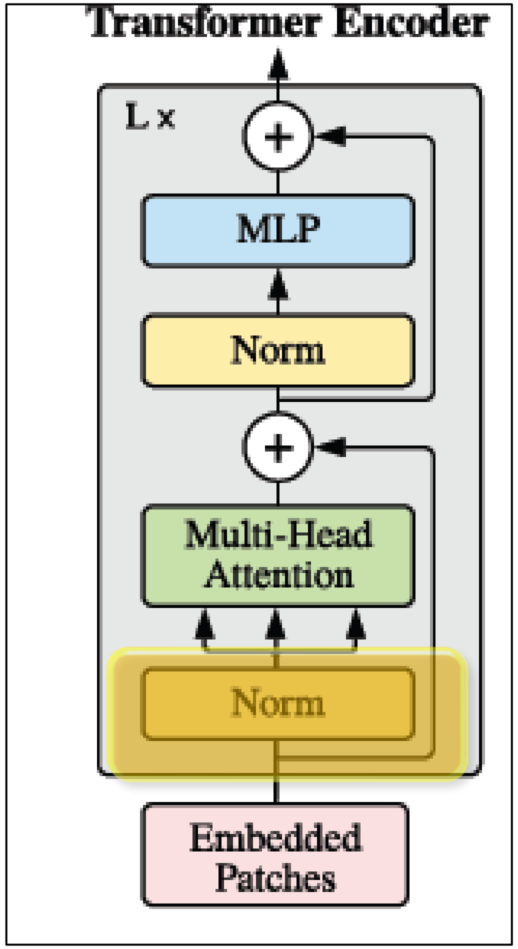
$$\mathrm{LN}(x) = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \cdot \gamma + \beta$$

- γ Scale
- β Shift

Where:

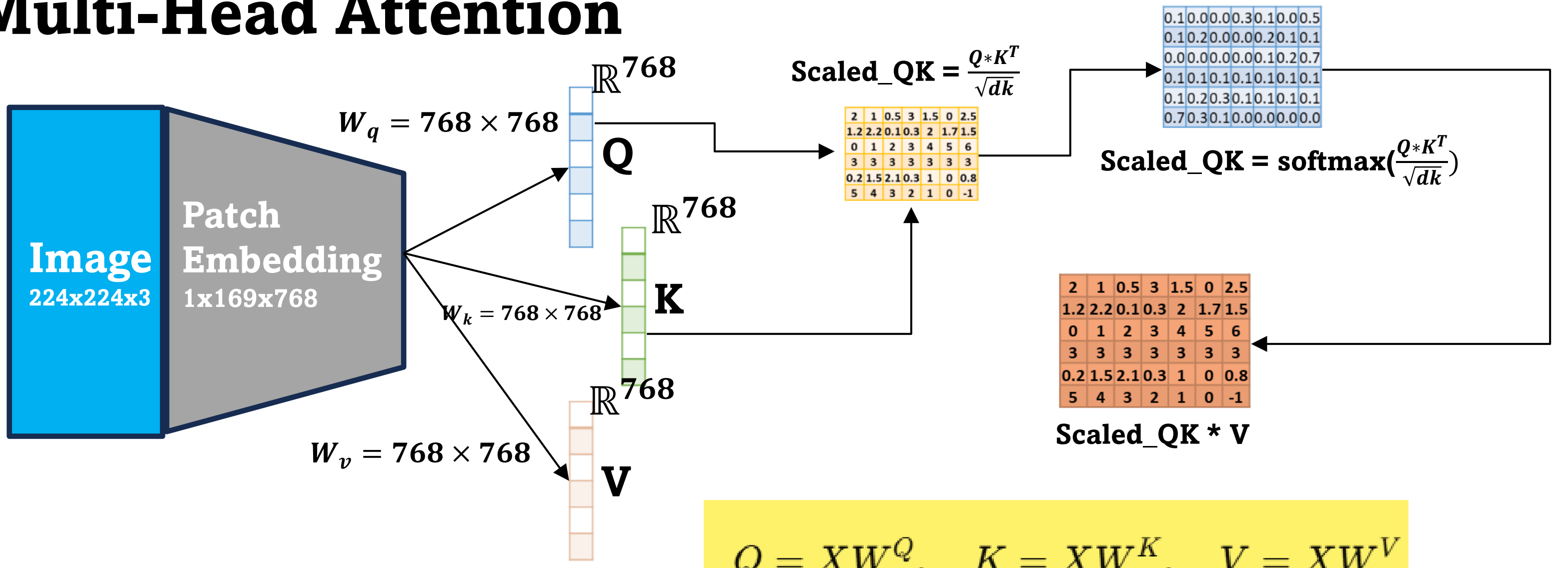
- $\mu = \frac{1}{d} \sum x_i, \sigma^2 = \frac{1}{d} \sum (x_i - \mu)^2$
- $\gamma, \beta \in \mathbb{R}^d$ are learnable parameters

```
layer_norm = nn.LayerNorm(emb_size)
x_norm = layer_norm(output_tensor) # B x N x D
```



Vision Transformer (ViT)

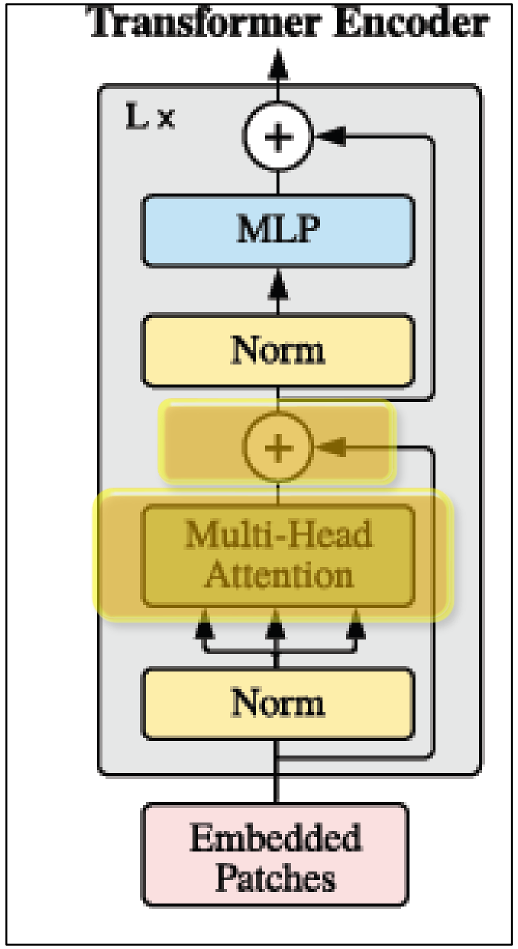
Multi-Head Attention



$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

Head = $\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$

$$\text{MultiHead}(X) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$




```
attn = nn.MultiheadAttention(embed_dim=emb_size, num_heads=8, batch_first=True)
x_attn, _ = attn(x_norm, x_norm, x_norm)
x = x + x_attn  # Residual connection # [1, 196, 768]
```

Vision Transformer (ViT)

Multi Layer Perceptron and Output Layer

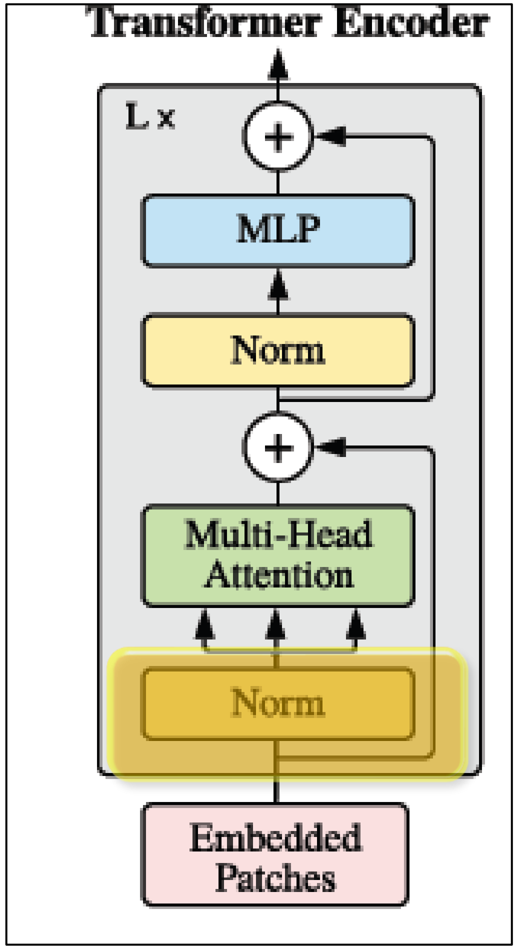
$$\text{MLP}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2$$

Usually, $W_1 \in \mathbb{R}^{D\times 4D}$, $W_2 \in \mathbb{R}^{4D\times D}$

```
x = layer_norm(x) # norm before feedforward # [1, 196, 768]

mlp = nn.Sequential(
    nn.Linear(emb_size, emb_size * 4),
    nn.ReLU(),
    nn.Linear(emb_size * 4, emb_size)
)
x_mlp = mlp(x)  # Feed forward network
x = x + x_mlp  # Residual connection # [1, 196, 768]
```

```
n_classes =  1000  # Number of classes for ImageNet
# Output head for ImageNet
output_layer = nn.Sequential(
    nn.Linear(emb_size, n_classes),      # from last hidden size to 1000 classes
    nn.Softmax(dim=1)          # softmax over class dimension
)
```



Multi Model LLM

Multi Model LLM

- A Multimodel LLM is a model that can process and reason across multiple data modalities — like text, images, audio, and video.
- The most common today are text-image models (like GPT-4V, Flamingo, Kosmos-2).
- Traditional LLMs only take in text as input. Multimodal LLMs extend this to also accept images (or even audio/video), enabling a richer form of understanding.

Why we require Multi Model LLM

- Broader understanding: Real-world tasks involve both visual and textual info (e.g., analyzing charts, understanding documents, describing scenes).
- Foundation for real-world apps: e.g., visual Q&A, medical imaging reports, video summarization.
- Paves the way for AGI: Human intelligence is multimodal — so models aiming to mimic it should be, too.

Before understanding the multimodal LLM, let's understand how transformers are used in images.

Multi Model LLM

Step 1: Encode image

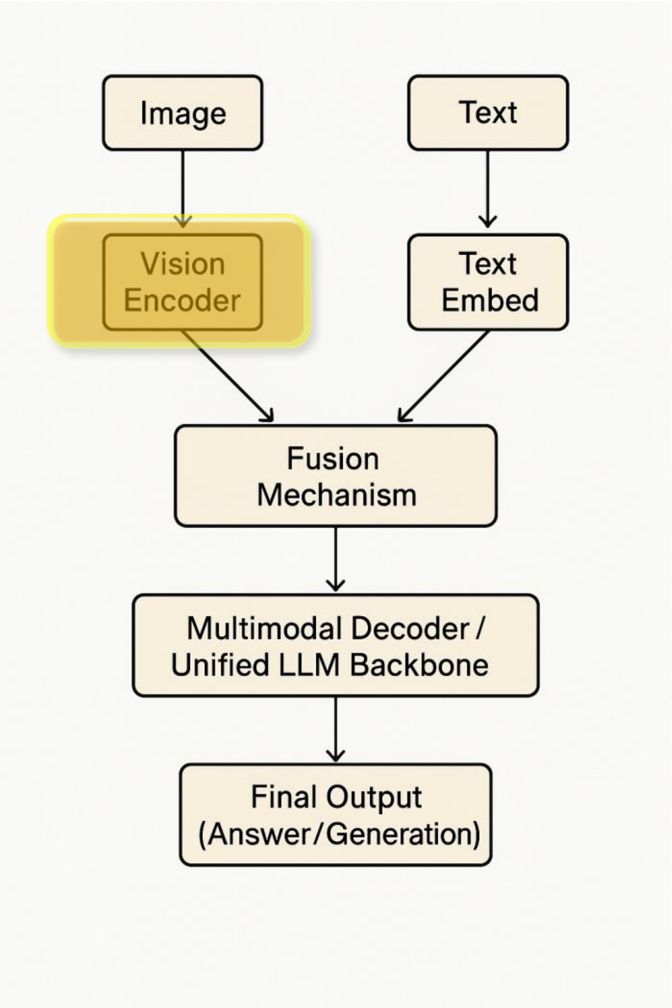
Image is divided into patches or regions and passed through a vision
ResNet). Each patch becomes a vector:

$$\mathbf{I} = [\vec{i}_1, \vec{i}_2, ..., \vec{i}_k] \in \mathbb{R}^{k \times d}$$

Where

k: number of image patches

d: hidden size



Multi Model LLM

Step 2: Encode Image with Positional Encoding

Patch tokens are embedded and positional encodings added:

$$I = [\overrightarrow{i_1}, \overrightarrow{i_2}, \overrightarrow{i_3}, \overrightarrow{i_4}, \dots, \overrightarrow{i_n}] \in \mathbb{R}^D$$

$$\overrightarrow{i_i} = \overrightarrow{i_i} + \overrightarrow{p_i}$$

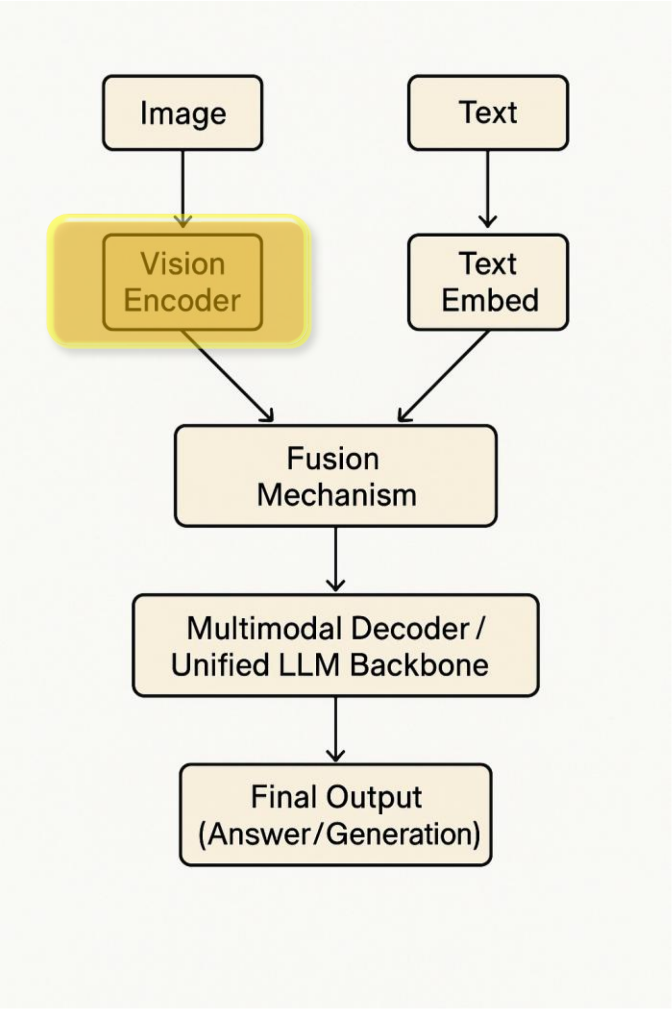
Where

$\overrightarrow{t_i}$ is the patch + positional encoding

$\overrightarrow{i_i}$ is the patch encoding

- Treats patches as a flat sequence.
- Uses learnable or sinusoidal encoding

$\overrightarrow{p_i}$ is the positional encoding



Multi Model LLM

Encode Text with Position Encoding

Text: A colorful bouquet to brighten up your day

First perform tokenization

Text Embedding = Token Embedding + Positional Encoding

Token Embedding

Each token is converted into a dense vector using a learned embedding layer.

Word → Index → Vector

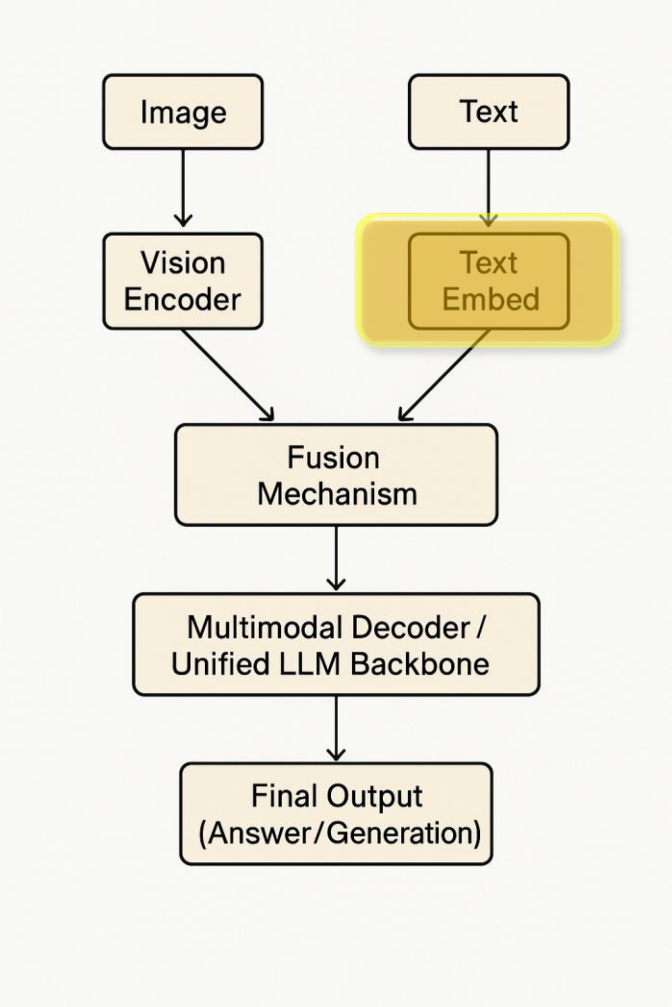
Positional Embedding

Each token is converted into a dense vector using a formula given below.

$$\mathbf{T} = [\vec{t}_1, \vec{t}_2, ..., \vec{t}_n] \in \mathbb{R}^{n \times d}$$

$$\mathrm{PE}(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{\mathrm{model}}}}}\right)$$

$$\mathrm{PE}(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{\mathrm{model}}}}}\right)$$



Multi Model LLM

Step-3: Fusion (Cross Attention)

Let's denote the image embeddings as key-value pairs, and the text embeddings as queries:

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

For **multimodal cross-attention**, say:

Query -> Text

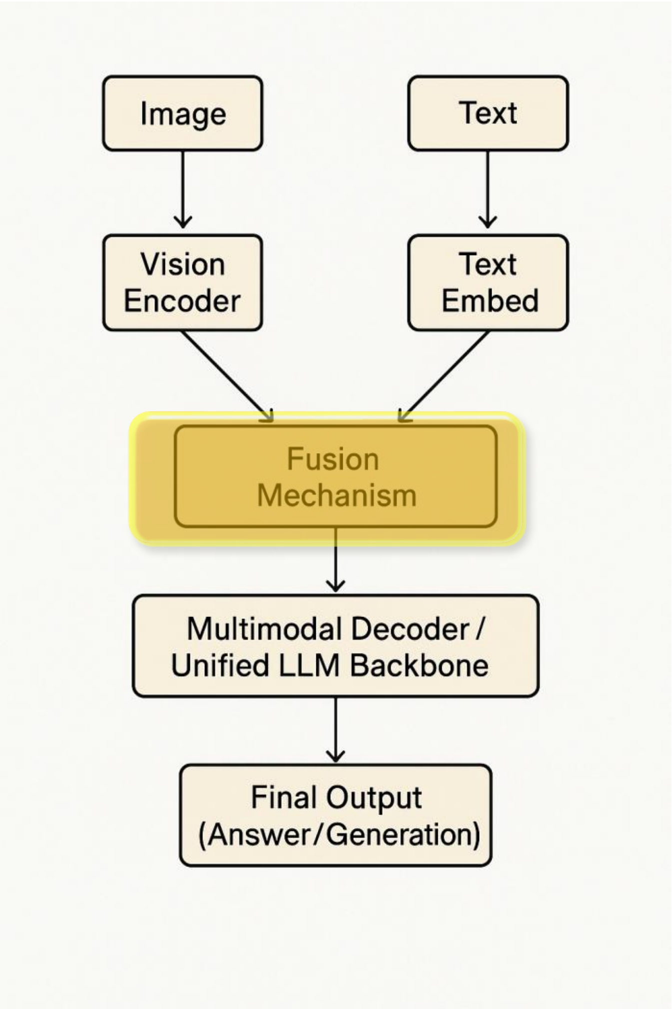
Key & Value -> Image

$$Q = W_Q \cdot \mathbf{T}$$

$$K, V = W_K \cdot \mathbf{I}, W_V \cdot \mathbf{I}$$

This lets the text query relevant visual patches and attend to the right image regions.

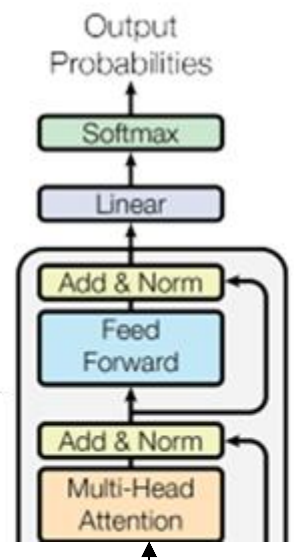
This is **text-to-image cross-attention**.

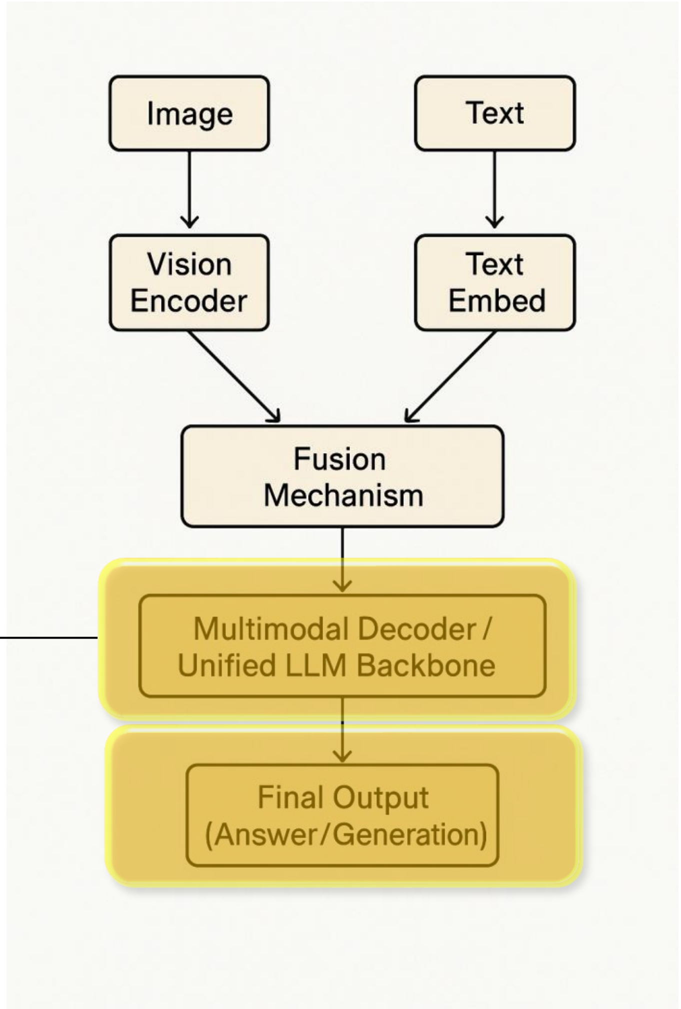


Multi Model LLM

Step-4: Multi-Modal Decoder

Single language model decoder (like GPT, LLaMA, Mistral) to generate text autoregressively, conditioned on a combination of image and text representations.





Property	**Value**
Type	**Autoregressive (Causal) Language Model**
Architecture	Transformer Decoder (like GPT)
Training Objective	Next token prediction (language modeling loss)
Token Generation	One token at a time (greedy, beam, sampling)
Output	Text (sequence of tokens decoded into words)

Vision-Language Models

- These are a class of multimodal models trained specifically to process visual and linguistic data together.
- Examples include:
  - CLIP: Contrastive model that learns to align text and image representations.
  - BLIP / Flamingo / GIT: Generative models trained to answer questions, generate captions, or follow multimodal prompts.

CLIP (Contrastive Language–Image Pretraining)

- CLIP is a Vision-Language model developed by OpenAI that learns to connect images and text in a shared embedding space using contrastive learning.
- Its key innovation is learning from natural language supervision, i.e., (image, caption) pairs, instead of predefined classification labels.

https://github.com/openai/CLIP
https://openai.com/index/clip/

Contrastive Language–Image Pretraining (CLIP)

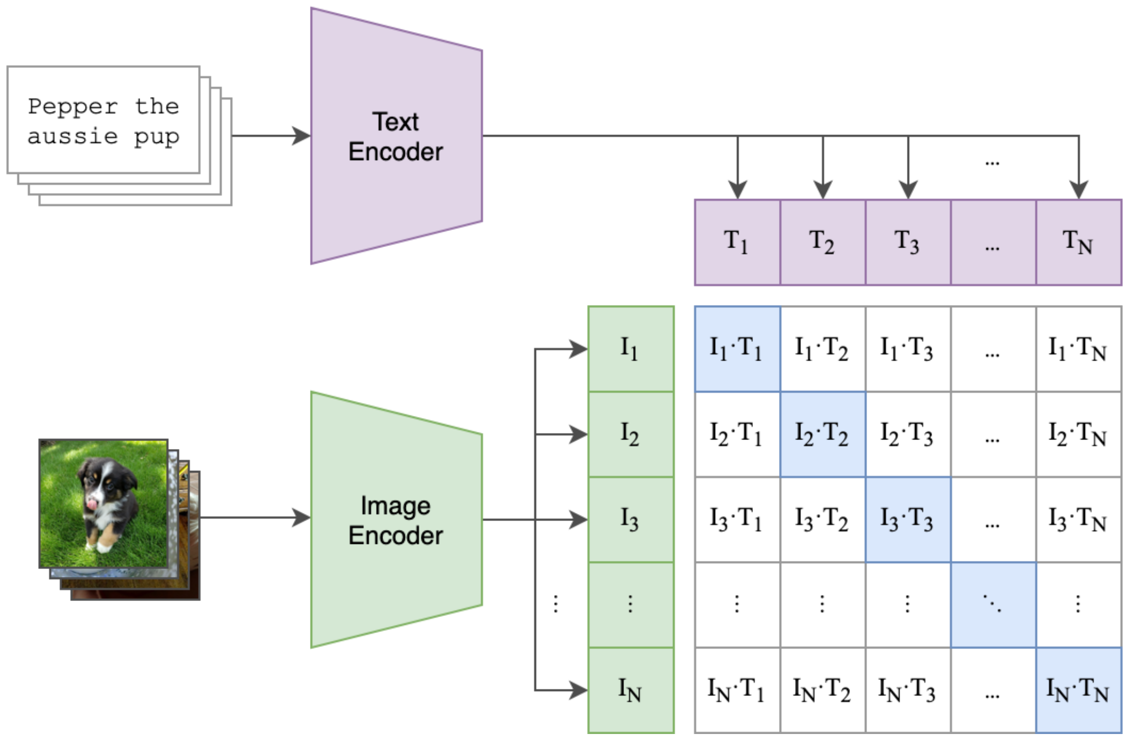
Image Encoder (ViT or ResNet)

- Takes an image and encodes it into a fixed-length vector.
- Examples: Vision Transformer (ViT-B/32) or ResNet-50.2.

Text Encoder (Transformer)

- Takes a text prompt (e.g., "a photo of a cat") and encodes it into a vector using a Transformer (like GPT-style).
- **Projection to Shared Embedding** SpaceBoth encoders map their outputs into the same latent space via projection heads (linear layers).

(1) Contrastive pre-training



https://github.com/openai/CLIP
https://openai.com/index/clip/

Contrastive Language–Image Pretraining (CLIP)

The image transformer output is in shape: $\mathbb{R}^{n\times D}$

→ n: number of image tokens (e.g., patches)

→ D: image embedding dimension.

n=50, D=768
→ image transformer outputs 50 patch embeddings of 768 dims
<u>Take a mean of all image patches or use CLS token embedding</u>

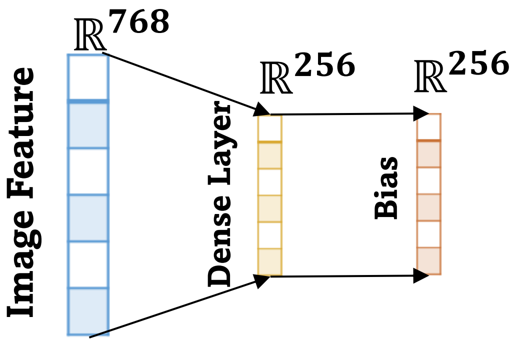
The text transformer output is in shape: $\mathbb{R}^{m\times P}$

→ mm: number of tokens in the text

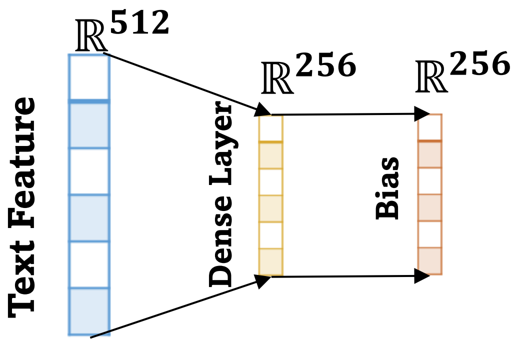
→ PP: text embedding dimension

m=12, P=512
→ text transformer outputs 12 token embeddings of 512 dims
<u>Take a mean of all image patches or use EOS token embedding</u>

Projection to Shared Embedding [let's say the dimension is 256]



Population
$I_{1\times256}$



Population
$T_{1\times256}$

Contrastive Language–Image Pretraining (CLIP)

Batch size = B

Image embeddings $\rightarrow B \times 256$

Text embeddings $\rightarrow B \times 256$

Cosine similarity matrix $\rightarrow B \times B$ $(S \in \mathbb{R}^{B \times B})$

Contrastive Learning

Let's say we have the following configurations

After finding the cosine similarity between all the pairs

We calculate **InfoNCE Loss** to encourage matching pairs to be close and others far apart.

Image → Text Loss

Treat each row i as a query, and column j as candidate text:

$$\mathcal{L}_{\text{image-to-text}} = -\frac{1}{B}\sum_{i=1}^{B}\log\frac{e^{S_{i,i}/\tau}}{\sum_{j=1}^{B} e^{S_{i,j}/\tau}}$$

Text → Image Loss

Treat each column j as a query, and row i as candidate image:

$$\mathcal{L}_{\text{text-to-image}} = -\frac{1}{B}\sum_{j=1}^{B}\log\frac{e^{S_{j,j}/\tau}}{\sum_{i=1}^{B} e^{S_{i,j}/\tau}}$$

Final Loss

$$\mathcal{L}_{\text{CLIP}} = \frac{1}{2}\left(\mathcal{L}_{\text{image-to-text}} + \mathcal{L}_{\text{text-to-image}}\right)$$

τ is the **temperature** parameter (a learnable or fixed scalar that sharpens or smoothens the distribution)

Contrastive Language–Image Pretraining (CLIP)

Contrastive Learning

Let's say similarity matrix S (after cosine similarity) is:

	T_1	T_2	T_3
I_1	2.0	0.5	0.3
I_2	0.2	2.5	0.4
I_3	0.1	0.6	3.0

For image-to-text (I_1 with T_1)

$$\mathcal{L}_1 = -\log\left(\frac{e^{2.0/\tau}}{e^{2.0/\tau} + e^{0.5/\tau} + e^{0.3/\tau}}\right)$$

For text-to-image (T_1 true pair is T_1)

$$\text{Logits} = [2.0, 0.2, 0.1] \quad (\mathrm{I}_1, \mathrm{I}_2, \mathrm{I}_3)$$

$$\text{Softmax Denominator} = e^{2.0} + e^{0.2} + e^{0.1} = 7.3891 + 1.2214 + 1.1052 \approx 9.7157$$

$$\text{Correct log-prob } (\mathrm{T}_1 \text{ with } \mathrm{I}_1) = -\log\left(\frac{e^{2.0}}{9.7157}\right) = -\log\left(\frac{7.3891}{9.7157}\right) \approx -\log(0.7607) \approx 0.274$$

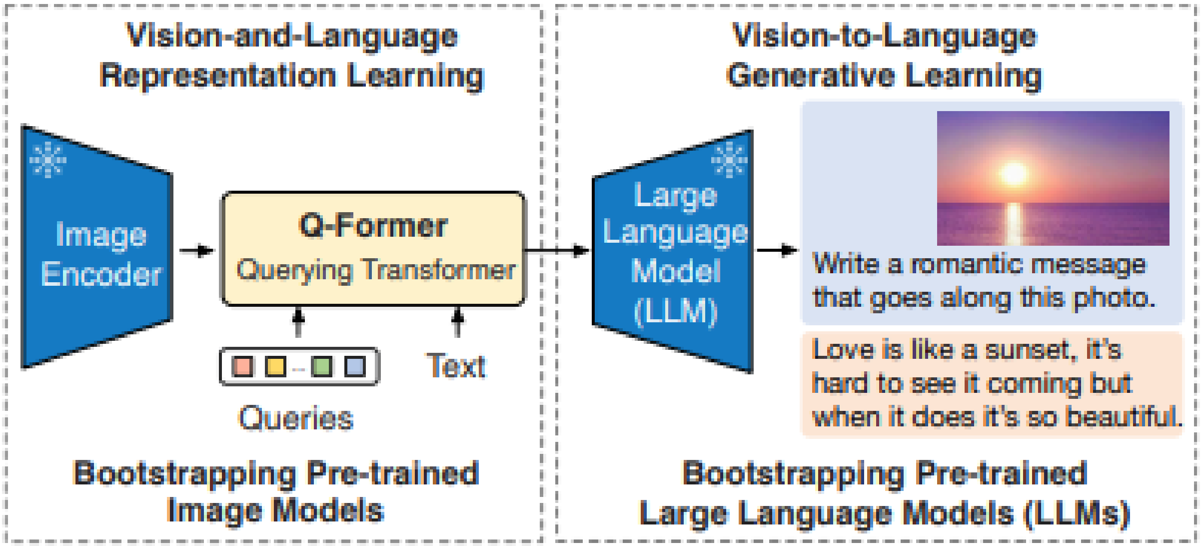
https://github.com/openai/CLIP
https://openai.com/index/clip/

BLIP (Bootstrapping Language-Image Pretraining)

BLIP (Bootstrapping Language-Image Pretraining) — a vision-language model architecture introduced by Salesforce Research in 2022.

It aims to learn a unified vision-language representation that can:

- Understand images and associated text (e.g., captions, questions).
- Generate or retrieve text based on visual input.



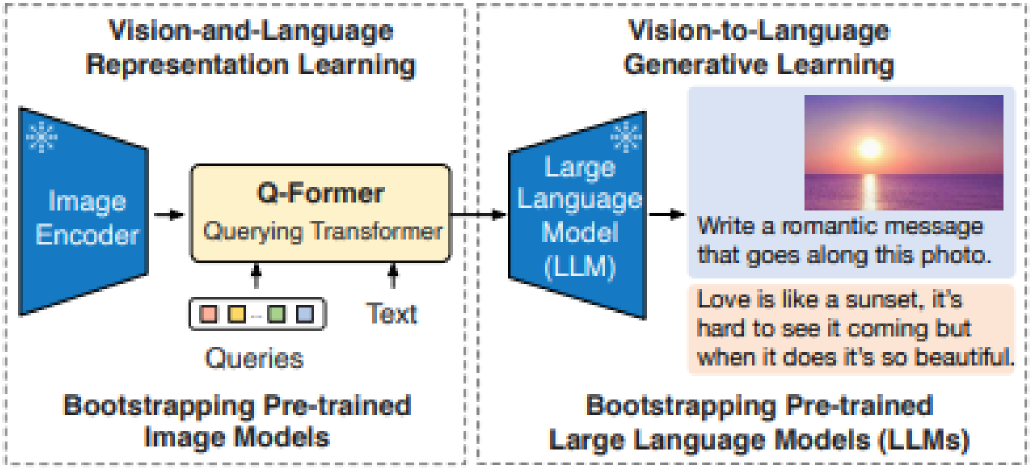
BLIP, https://arxiv.org/pdf/2201.12086
BLIP2, https://arxiv.org/pdf/2301.12597

BLIP (Bootstrapping Language-Image Pretraining)

Image Encoder

Typically, a Vision Transformer (ViT) or ResNet, it encodes images into patch-based embeddings.

- Input: Image
- Output: Visual embeddings V $\in \mathbb{R}^{n\times d}$



Text Encoder

A BERT-like transformer that encodes input text.

Input: Tokenized sentence like "a cat sitting on the couch"

Output: Token embeddings T $\in \mathbb{R}^{m\times d}$

BLIP, https://arxiv.org/pdf/2201.12086
BLIP2, https://arxiv.org/pdf/2301.12597

BLIP (Bootstrapping Language-Image Pretraining)

Vision-Language Fusion Encoder (Q-Former)

Let's say

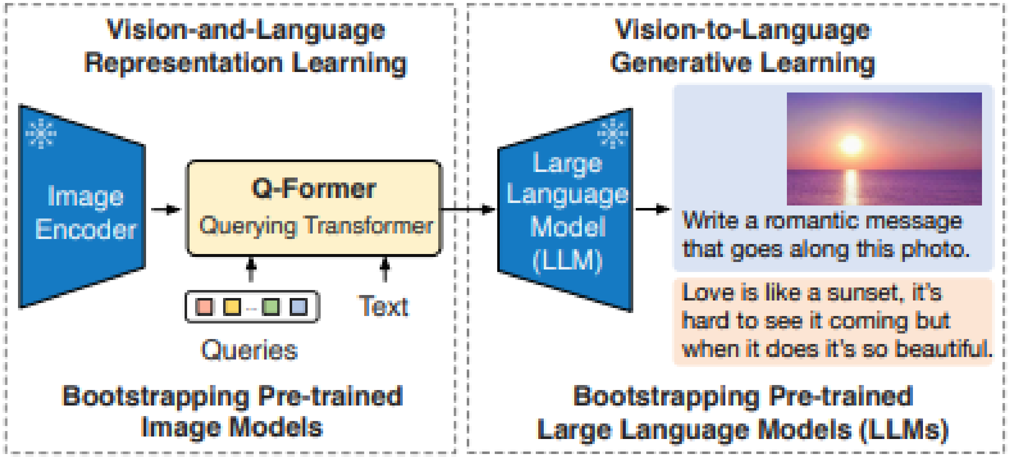
- Image embedding: $V \in \mathbb{R}^{n \times d}$
- Text embedding: $T \in \mathbb{R}^{m \times d}$
- Query tokens: $Q \in \mathbb{R}^{k \times d}$ *where* $k \ll n$

Q learns to extract or summarize relevant semantic information from T (and V) via attention.

$$K = [V; T] \in \mathbb{R}^{(n+m) \times d}$$

During cross-attention, each query token $q_i \in Q$ attends to all the keys and values in K:

$$\text{Attention}(q_i, K, K) = \sum_j \text{softmax}(q_i^\top K_j / \sqrt{d}) \cdot K_j$$



This means:
Each query token extracts relevant context from the entire image + text space.
It lets the model jointly reason over both modalities.

https://arxiv.org/pdf/2201.12086

BLIP (Bootstrapping Language-Image Pretraining)

LLM as Text Generator

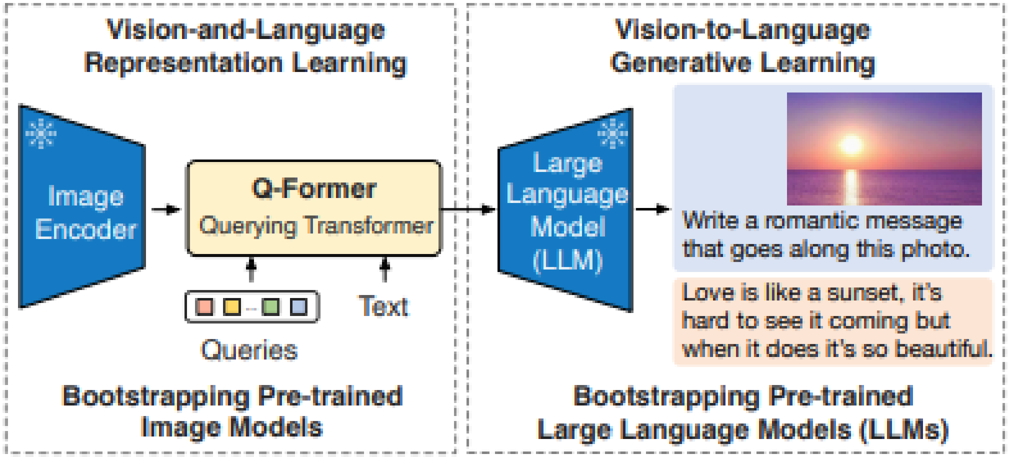
The LLM takes as input the visual understanding produced by the Q-Former (a summary of image features), and generates natural language text.

For example, given a sunset image, it might generate: “Love is like a sunset…”

Leverages Pretrained Language Knowledge

- The LLM (e.g., OPT, GPT, or LLaMA) is pretrained on massive text corpora.
- BLIP-2 bootstraps this pretrained LLM, instead of training from scratch.
- This means the model already knows:
  - Grammar and fluency
  - World knowledge
  - Common-sense reasoning

So when the Q-Former encodes "sunset, sky, reflection", the LLM knows how to turn that into poetic or task-specific language.



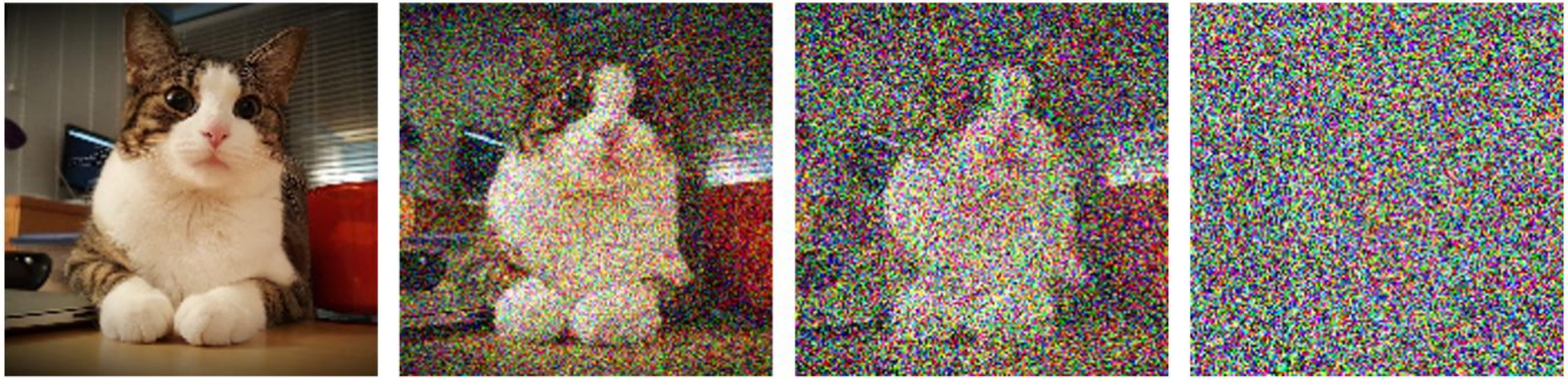
https://arxiv.org/pdf/2201.12086

Diffusion Models

Diffusion Models

- A diffusion model is a generative model that learns to create data (like images) by reversing a gradual noise-adding process.
- Inspired by thermodynamics, these models start with pure noise and learn how to denoise it step-by-step to generate realistic data.

Forward Process (Diffusion): Gradually add Gaussian noise to a data sample.



Reverse Process (Denoising): Learn to reverse the noise and recover the data.

Diffusion Models

Forward Process (q)

Given a real data sample x_0, we define a Markov chain:

$$q(x_t \mid x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t\mathbf{I})$$

Where,

- x_0 is the original image.
- x_1 , x_2 , x_3 ,…, x_T are noisy versions.
- β_t is a small variance schedule (e.g., linear or cosine).
- At $t = T$, x_T becomes close to pure noise.
- I represents Identity matrix (means noise is added isotropically)

Diffusion Models

$$q(x_t \mid x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t\mathbf{I})$$

Forward Process (q)

Let's Generate (x_1) from (x_0) $\quad x_1 = \sqrt{1-\beta_1} \cdot x_0 + \epsilon_1, \quad \epsilon_1 \sim \mathcal{N}(0, \beta_1\mathbf{I})$

Term	Meaning	Type / Shape
x_0	The **original image**	Tensor: [C×H×W], e.g., [3×224×224]
β_1	Noise level at step 1	Scalar (e.g., 0.0001)
$\sqrt{1-\beta_1}$	**Scaling factor** for image	Scalar
$\in_1$	**Gaussian noise**	Same shape as x_0
I	Identity covariance (used for isotropic noise)	Implied as diagonal matrix
x_1	Noisy version of image at step 1	Same shape as x_0

$$x_1 = \left(\sqrt{1-0.0001}\right)x_0 + \in_1$$

Each element in $\in_1$ is **independently sampled** from a **Gaussian distribution**:

$\epsilon_1 \in \mathbb{R}^{3\times224\times224}$ $\quad \epsilon_1[i,j,c] \sim \mathcal{N}(0, \beta_1)$ $\quad \sigma = \sqrt{0.0001} = 0.01$

Diffusion Models

We train a neural network to **reverse** the noise.

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

Given a noisy image x_t , we predict a less noisy version x_{t-1}, using a neural network.

We assume:

- The distribution of x_{t-1} given x_t is a Gaussian.
- We learn the mean and optionally the variance using a neural network.

Symbol	Meaning	Description / Value
θ	Learnable parameters	Weights of the neural network (usually U-Net)
$\mu_\theta(x_t, t)$	Predicted mean for x_{t-1}	Output of the neural network
$\Sigma_\theta(x_t, t)$	Predicted variance (optional)	Often fixed or simplified in practice

- μ_θ tells us: Where should x_{t-1} be centered around?
- Σ_θ tells us: How much uncertainty (noise) should be in the sample?

Latent Diffusion Models (LDM)

LDMs run diffusion in a latent space (compressed version of image space), not pixel space. This speeds up training and reduces memory.

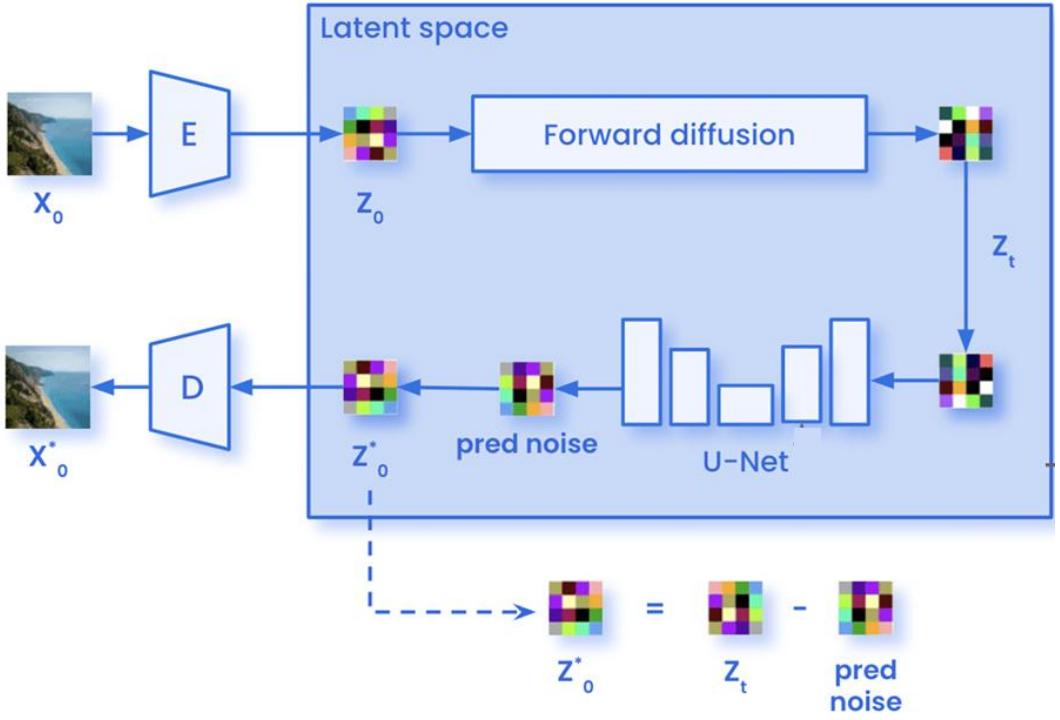
- **Efficient:** 10×–100× faster and less memory-intensive.
- **High quality:** Retains visual details via decoder.

- **Step-1:** Compress image using VAE encoder:

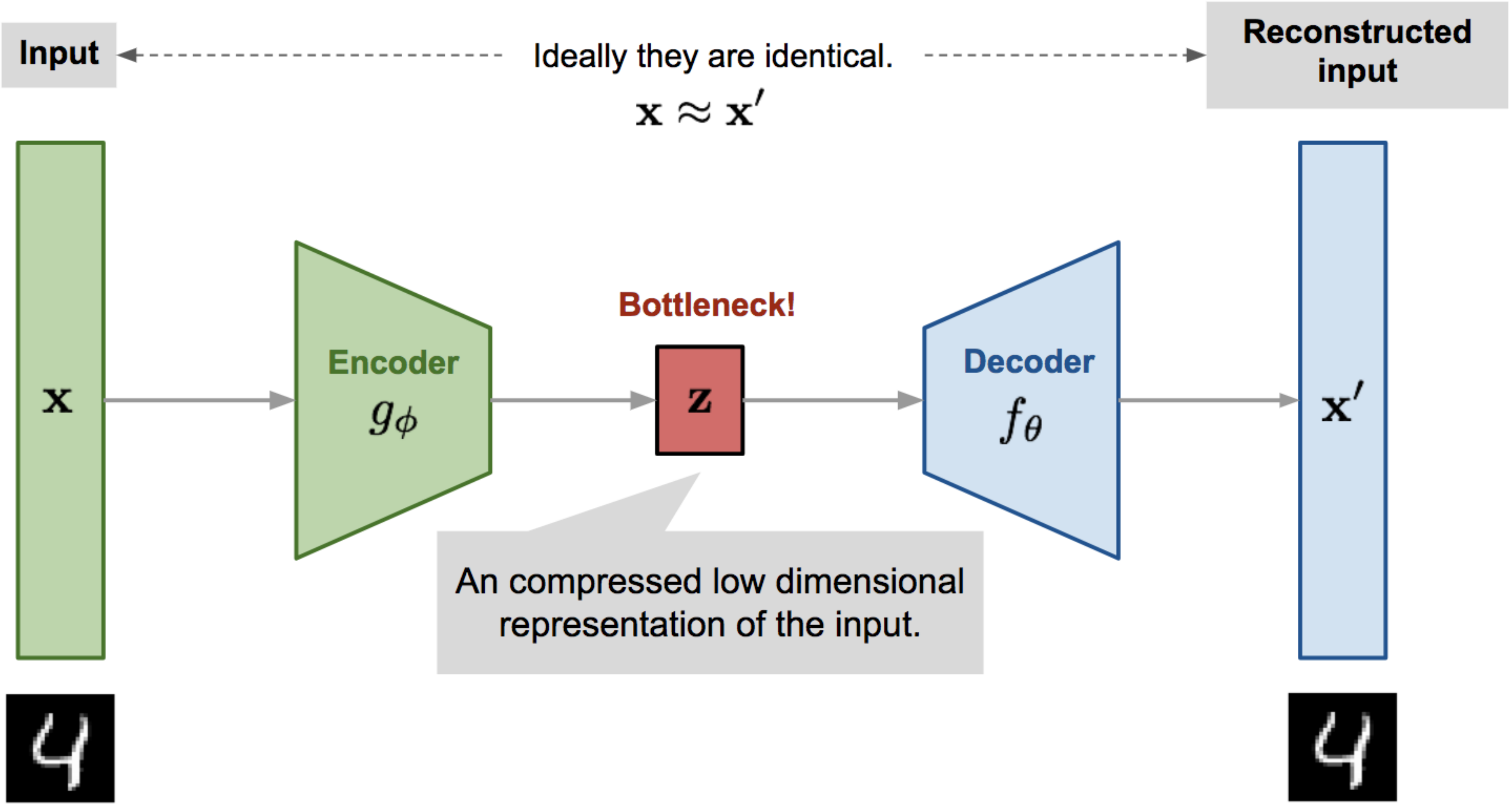
$$z = \text{Encoder}(x)$$

- **Step-2:** Apply diffusion on latent z_t instead of x_t.
- **Step-3:** After denoising, decode back to image:

$$x = \text{Decoder}(z)$$

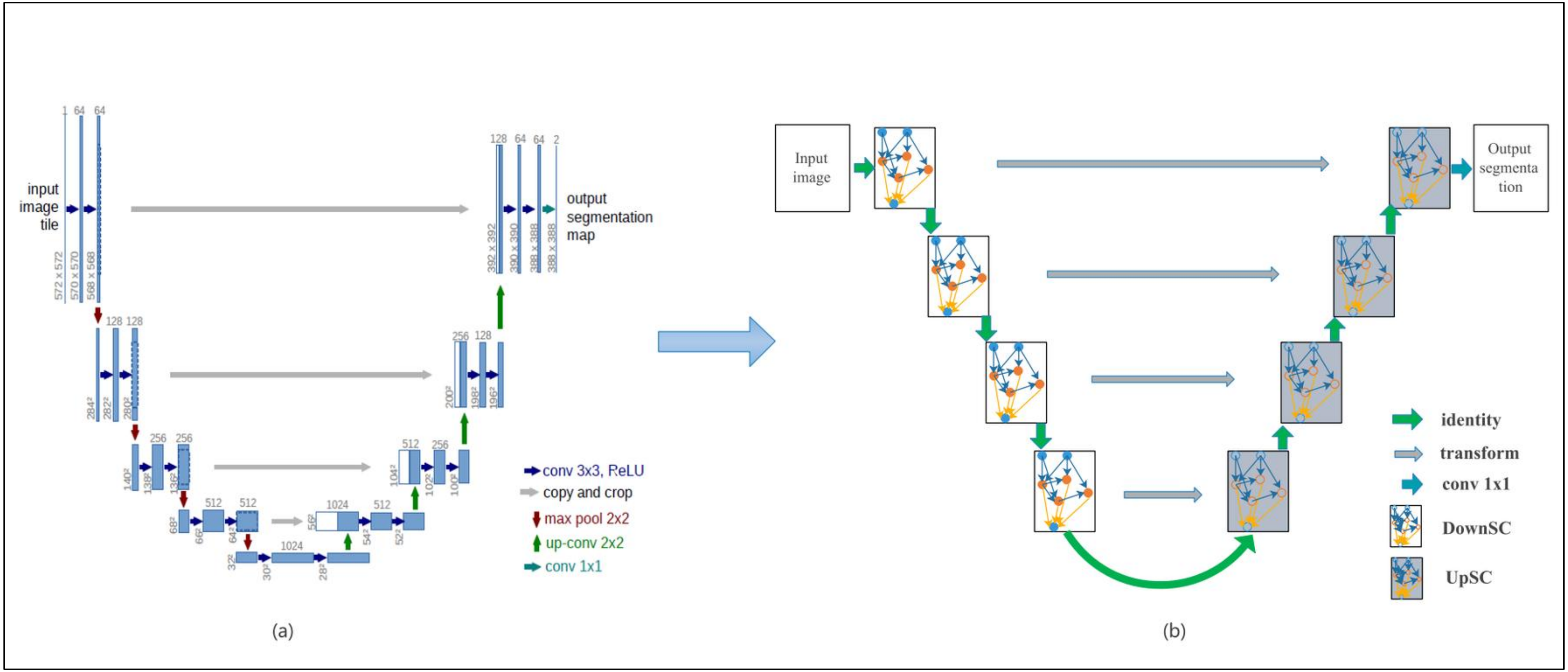


Variational Auto Encoder



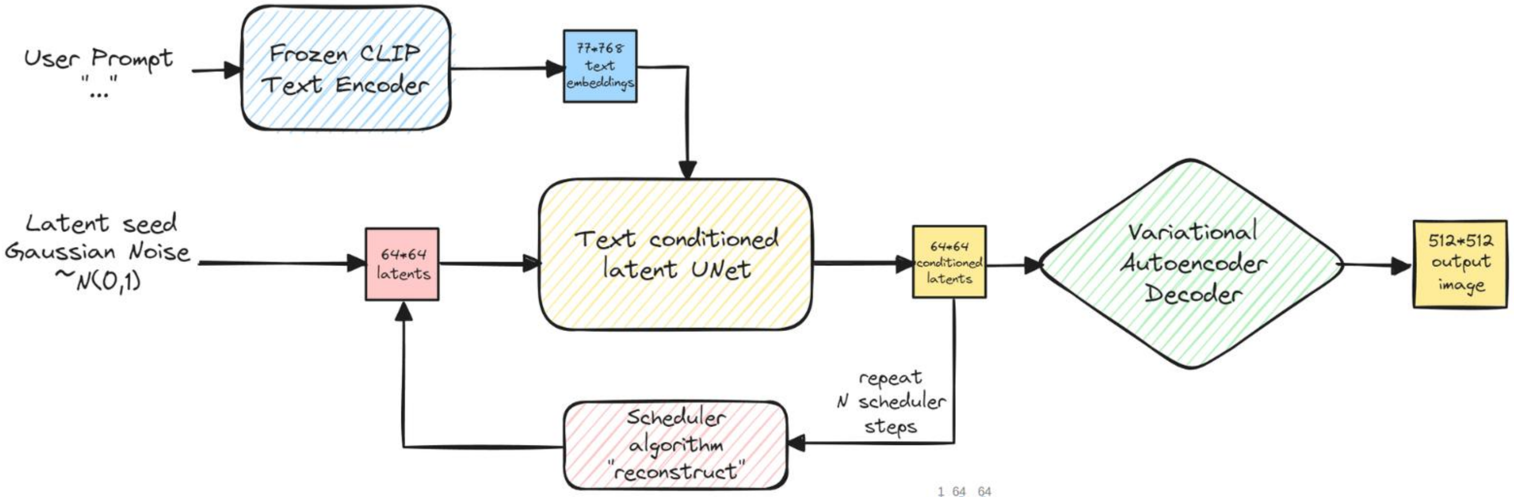
Kingma, Diederik P., and Max Welling. "Auto-encoding variational bayes." 20 Dec. 2013,

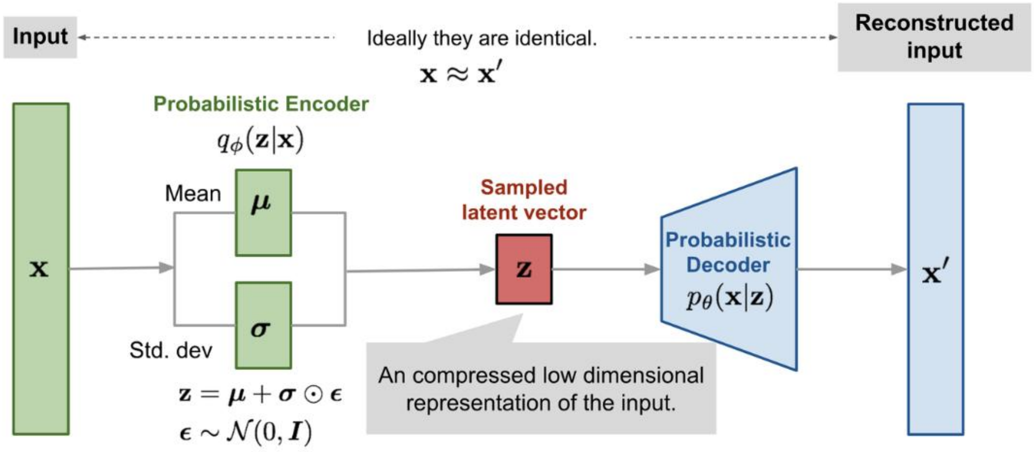
UNet

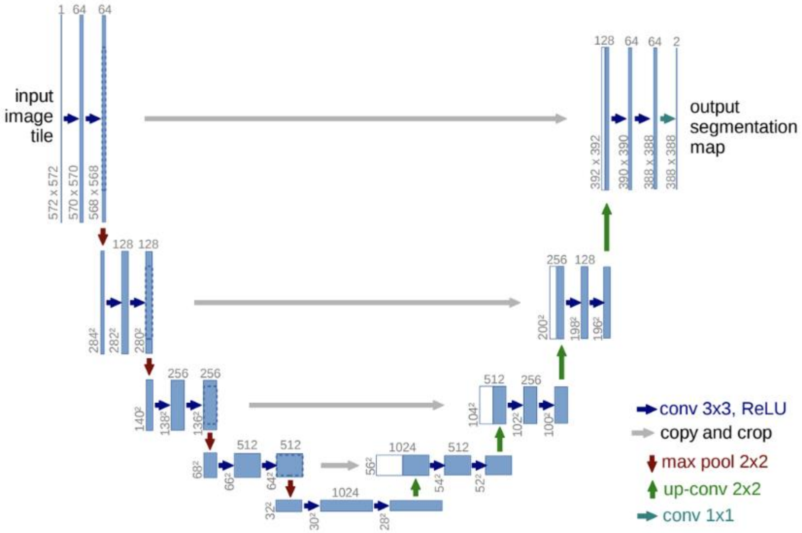


Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer international publishing, 2015.

Stable Diffusion







Rombach, Robin, et al. "High-resolution image synthesis with latent diffusion models." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022.

Thanks for your time