image segmentation prompt

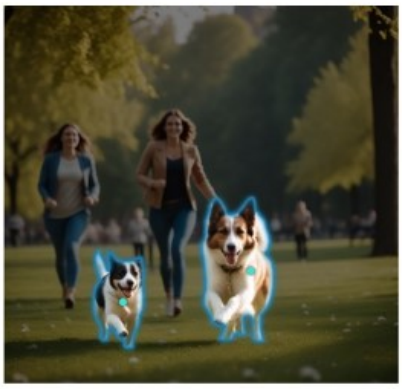
image_segmentation_prompt

juin 29, 2024

Introduction

+

User request:
"highlight dogs in the picture"

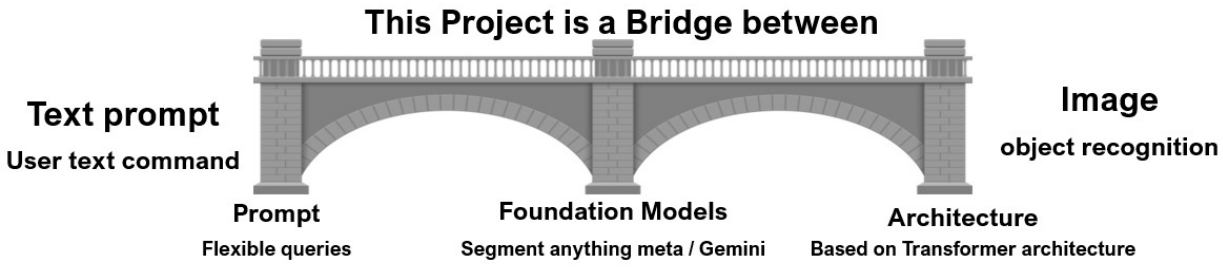


Output result

CHAPITRE 1

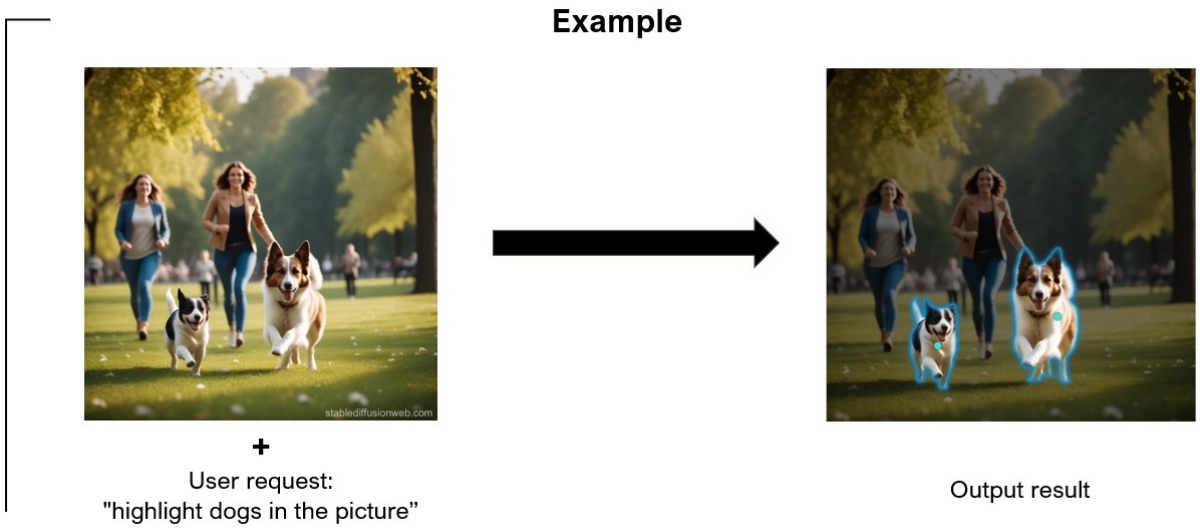
Project Introduction

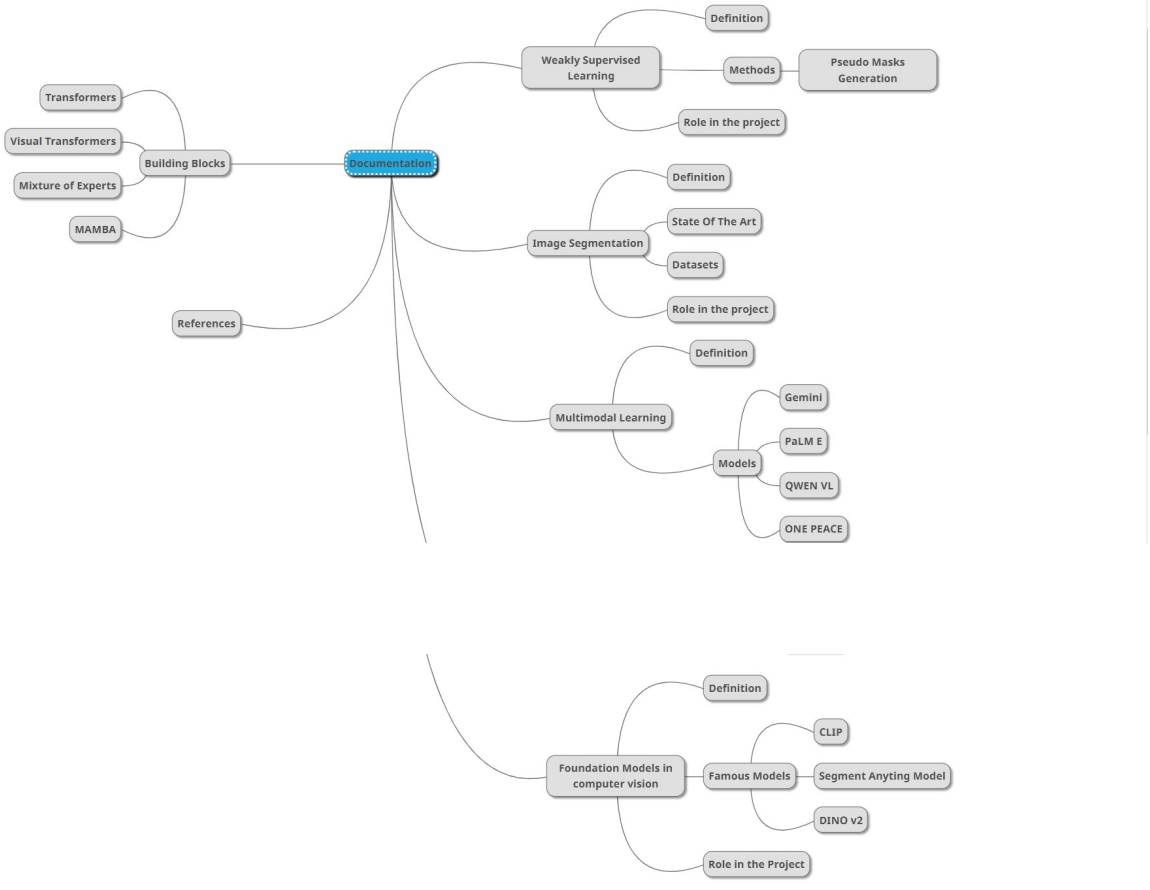
1.1 image segmentation prompt



1.2 Documentation axes

PyTorch


Example
stablediffusionweb.com
+
User request:
"highlight dogs in the picture"
Output result


Documentation
Transformers
Visual Transformers
Mixture of Experts
MAMBA
Building Blocks
References
Definition
Weakly Supervised Learning
Methods
Pseudo Masks Generation
Role in the project
Definition
State Of The Art
Datasets
Role in the project
Image Segmentation
Definition
Gemini
PaLM E
QWEN VL
ONE PEACE
Models
Multimodal Learning
Definition
CLIP
Segment Anyting Model
DINO v2
Famous Models
Role in the Project
Foundation Models in computer vision

CHAPITRE 2

PyTorch

2.1 1. Introduction

Purpose

The purpose of this documentation is to provide a comprehensive introduction to tensors in PyTorch,emphasizing their importance and usage within the context of machine learning models.

2.2 2. Installing PyTorch with Anaconda

` conda create --name pytorch_env `

` conda activate pytorch_env `

` conda install pytorch torchvision cpuonly -c pytorch `

` conda install pytorch torchvision cudatoolkit -c pytorch `

` python -c "import torch; print(torch.__version__)" `

2.3 3. Introduction to Tensors

```python
import torch

# Create a tensor
x = torch.tensor([[1, 2], [3, 4]])
print(x)
```

— **Initializing Tensors**

```
import torch
import numpy as np

# Initialize from data
data = [[1, 2], [3, 4]]
x_data = torch.tensor(data)

# Initialize from NumPy array
np_array = np.array(data)
x_np = torch.from_numpy(np_array)

print(x_data)
print(x_np)
```

— **Attributes of Tensors**

```
import torch

# Create a tensor
tensor = torch.rand(3, 4)

# Get tensor attributes
print(f"Shape of tensor: {tensor.shape}")
print(f"Datatype of tensor: {tensor.dtype}")
print(f"Device tensor is stored on: {tensor.device}")
```

— **Operations on Tensors**

```
import torch

# Arithmetic operations
x = torch.tensor([[1, 2], [3, 4]])
y = torch.tensor([[5, 6], [7, 8]])

# Matrix multiplication
z1 = x @ y
z2 = torch.matmul(x, y)

print(z1)
print(z2)
```

— **Bridge with NumPy**

```
import torch
import numpy as np

# Tensor to NumPy array
tensor = torch.tensor([1, 2, 3, 4])
numpy_array = tensor.numpy()

# NumPy array to Tensor
numpy_array = np.array([5, 6, 7, 8])
tensor = torch.from_numpy(numpy_array)
```

(suite sur la page suivante)

(suite de la page précédente)

```
print(tensor)
```

Note : For more practice and to learn more, we can visit this tutorial.

Find the link to github repository

Find the link to colab

2.4 4. Datasets & DataLoaders

— Dataset :

— DataLoader :

— **Loading a Dataset**

```
import torch
from torch.utils.data import Dataset
from torchvision import datasets
from torchvision.transforms import ToTensor
import matplotlib.pyplot as plt


training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)

test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor()
)
```

— **Iterating and Visualizing the Dataset**

```
labels_map = {
    0: "T-Shirt",
    1: "Trouser",
    2: "Pullover",
    3: "Dress",
    4: "Coat",
    5: "Sandal",
    6: "Shirt",
    7: "Sneaker",
    8: "Bag",
    9: "Ankle Boot",
```

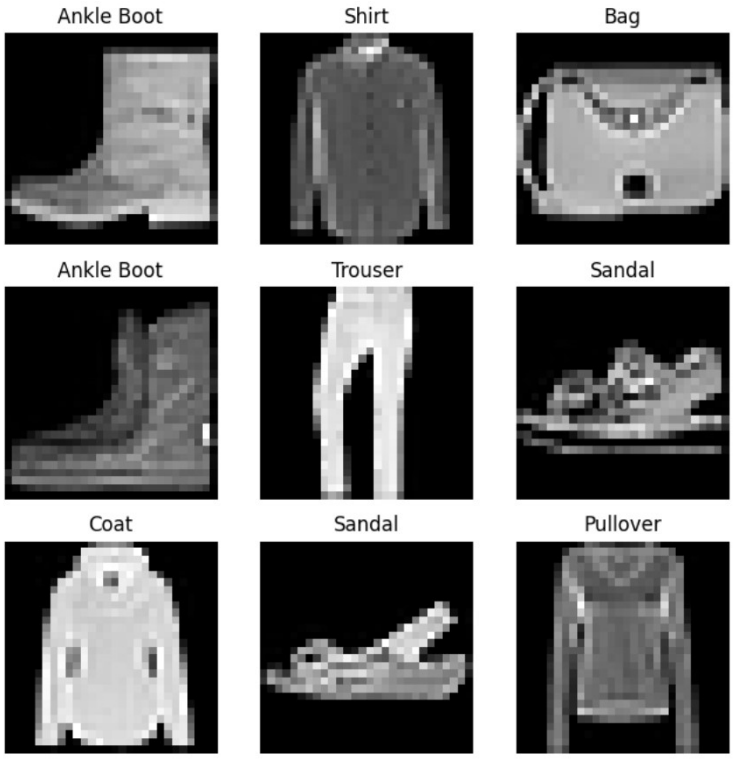
(suite sur la page suivante)

(suite de la page précédente)

```
    }
figure = plt.figure(figsize=(8, 8))
cols, rows = 3, 3
for i in range(1, cols * rows + 1):
    sample_idx = torch.randint(len(training_data), size=(1,)).item()
    img, label = training_data[sample_idx]
    figure.add_subplot(rows, cols, i)
    plt.title(labels_map[label])
    plt.axis("off")
    plt.imshow(img.squeeze(), cmap="gray")
plt.show()
```

— output

This code generates a grid of images with their corresponding labels from the Fashion-MNIST dataset. Each image represents a clothing item, and the labels indicate the category of the clothing.



— **Creating a Custom Dataset for Your Files**

```
import os
import pandas as pd
from torchvision.io import read_image
from torch.utils.data import Dataset

class CustomImageDataset(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None, target_transform=None):
        self.img_labels = pd.read_csv(annotations_file)
```

(suite sur la page suivante)

(suite de la page précédente)

```
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
        image = read_image(img_path)
        label = self.img_labels.iloc[idx, 1]
        if self.transform:
            image = self.transform(image)
        if self.target_transform:
            label = self.target_transform(label)
        return image, label
```

__init__

__len__

Example :

```
def __len__(self):
    return len(self.img_labels)
```

__getitem__

Example :

```
def __getitem__(self, idx):
    img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
    image = read_image(img_path)
    label = self.img_labels.iloc[idx, 1]
    if self.transform:
        image = self.transform(image)
    if self.target_transform:
        label = self.target_transform(label)
    return image, label
```

— **Preparing Your Data for Training with DataLoaders**

```
from torch.utils.data import DataLoader

train_dataloader = DataLoader(training_data, batch_size=64, shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=64, shuffle=True)
```

— **Iterate Through the DataLoader**

Example :

```
# Display image and label.
train_features, train_labels = next(iter(train_dataloader))
print(f"Feature batch shape: {train_features.size()}")
print(f"Labels batch shape: {train_labels.size()}")
```

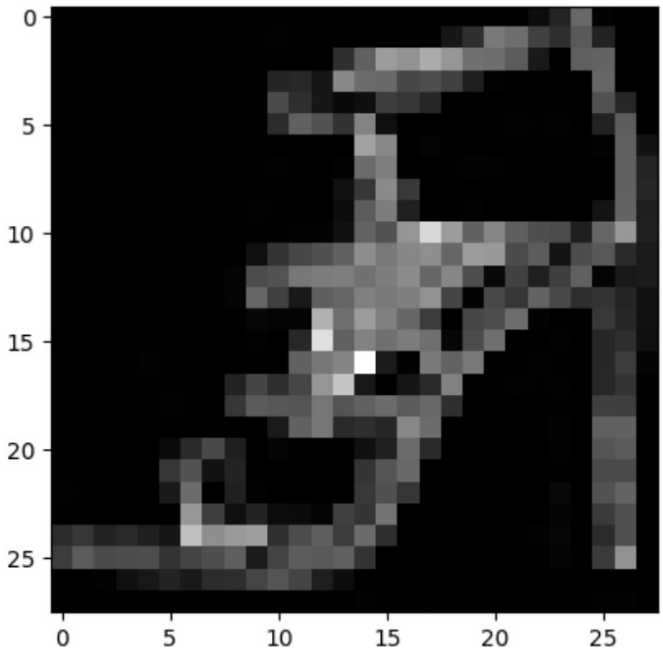
(suite sur la page suivante)

(suite de la page précédente)

```
img = train_features[0].squeeze()
label = train_labels[0]
plt.imshow(img, cmap="gray")
plt.show()
print(f"Label: {label}")
```

— output

This code segment outputs a batch of training features and their corresponding labels from the train_dataloader.



Note : For more practice and to learn more, we can visit this tutorial.

Find the link to Github repository

Find the link to colab

CHAPITRE 3

Neural Network

3.1 1. Introduction

In this article, we will build a neural network from scratch and use it to classify

3.2 2. dataset :

```
import pandas as pd
df = pd.read_csv('cancer_classification.csv')
```

Note : You can view the dataset and access by clicking the link to the dataset

Read the dataset :

3.3 3. Training on a simple dataset

1. DataPreprocessing :

Note : You can view the code and access by clicking the. link to DataPreprocessing class

2. DataExploration :

Note :

You can view the code and access by clicking the.
link to the DataExploration class

3. ModelTraining :

Note : You can view the code and access by clicking the link to the ModelTraining class.

4. ModelEvaluation :

Note : You can view the code and access by clicking the link to the ModelEvaluation class

5. NeuralNetwork :

Note : "You can view the code and access by clicking the link to the NeuralNetwork class.

3.4 4. Test the DataPreprocessing class

The *preprocessor* object is created using the DataPreprocessing class, which prepares the data for training a machine learning model. After splitting the data into training and testing sets using the split_data() method, it normalizes the data with normalize_data(). Finally, it converts the data into tensors with tensorize_data(), ready for model training and evaluation.

```
preprocessor = DataPreprocessing(df)
x_train, x_test, y_train, y_test = preprocessor.split_data(test_size=0.2, random_
↪state=42)
x_train, x_test = preprocessor.normalize_data()
x_train_tensor, x_test_tensor, y_train_tensor, y_test_tensor = preprocessor.tensorize_
↪data()
```

3.5 5. test the DataExploration class :

— **information_help()** : Their role is to display the methods existing in the DataExploration class.

```
explorer = DataExploration(df)
explorer.information_help()
```

output :

— **DisplayData()** : Displays the head of the DataFrame.

```
explorer = DataExploration(df)
print("DataFrame Head")
explorer.DisplayData()
```

— **DisplayDataTypes()** : Displays the data types of columns in the DataFrame.

```
print("\nData Types")
explorer.DisplayDataTypes()
```

— **DisplayDataInfo()** : Displays general information about the DataFrame.

```
print("\nData Info")
explorer.DisplayDataInfo()
```

— **DisplayDataDescription()** : Displays statistical descriptions of the data.

```
print("\nData Description")
explorer.DisplayDataDescription()
```

— **DisplayDataShape()** :Displays the shape of the DataFrame.

```
print("\nData Shape")
explorer.DisplayDataShape()
```

— **DisplayMissingValues()** :Displays information about missing values in the DataFrame.

```
print("\nMissing Values")
explorer.DisplayMissingValues()
```

— **DisplayCorrelationMatrix()** :Displays the correlation matrix of numerical features in the DataFrame.

```
print("\nCorrelation Matrix")
explorer.DisplayCorrelationMatrix()
```

— **DisplayCorrelationWithColumn("benign_0__mal_1")** :Displays the correlation of all features with the target column named "benign_0__mal_1".

```
print("\nCorrelation with 'target' column:")
explorer.DisplayCorrelationWithColumn('benign_0__mal_1')
```

— **DisplayHeatMap()** :Displays a heatmap of the correlation matrix.

```
print("\nHeatMap")
explorer.DisplayHeatMap()
```

3.6 5. test the NeuralNetwork class

```
input_features = len(df.columns) - 1
out_features = df['benign_0__mal_1'].unique().sum()
neural_net = NeuralNetwork(input_features, out_features)
print("Neural Network Architecture:")
print(neural_net)
```

output :

Neural Network Architecture :

```
NeuralNetwork(
(fc1): Linear(in_features=30, out_features=30, bias=True)
(fc2): Linear(in_features=30, out_features=15, bias=True)
(fc3): Linear(in_features=15, out_features=1, bias=True)
(relu): ReLU()
(sigmoid): Sigmoid()
)
```

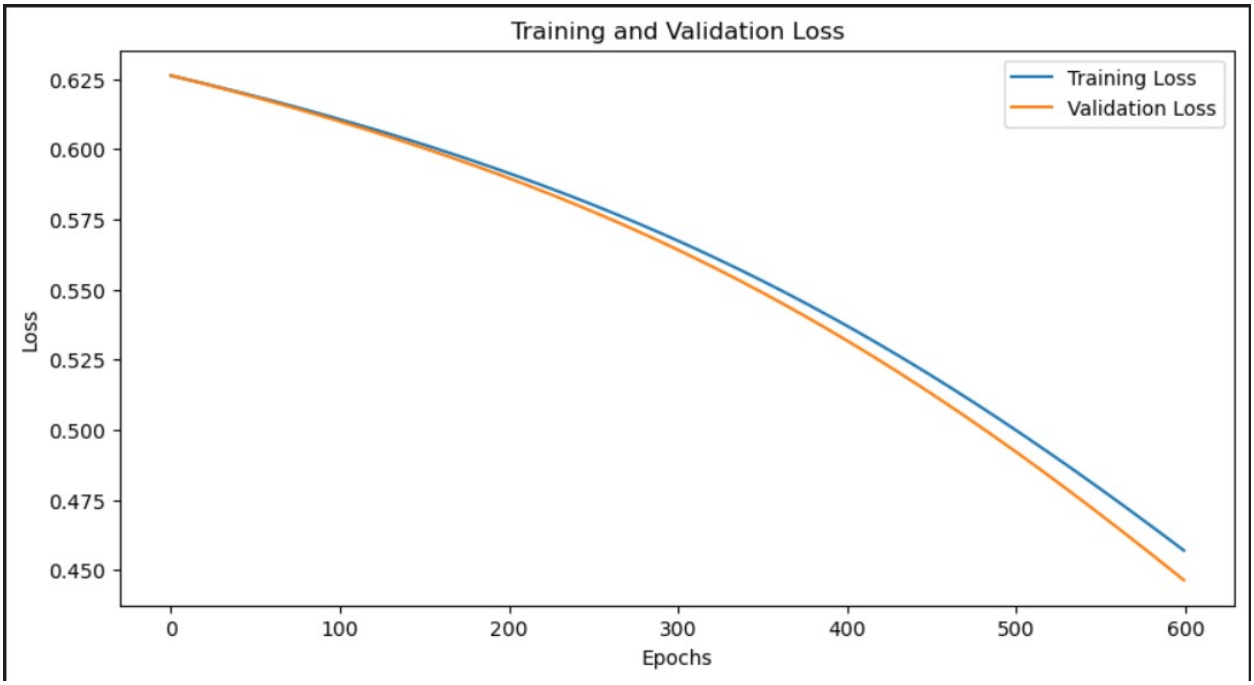
Here's the explanation :

3.7 6. Testing the ModelTraining class

This code snippet demonstrates setting up the neural network model, defining the loss function and optimizer, and then training the model using a ModelTrainer class. During training, it collects the training and testing losses for each epoch.

```
from torch import nn
model = neural_net
criterion = nn.BCELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
from modeltrainer import ModelTrainer
trainer = ModelTrainer(model, criterion, optimizer)
train_losses, test_losses = trainer.train(x_train_tensor, y_train_tensor, x_test_tensor,␣
↪y_test_tensor, epochs=600)
```

plot train_losses and test_losses



3.8 7. test the ModelEvaluation class

```
evaluator = ModelEvaluation(model, criterion, optimizer)
```

```
model.eval()
with torch.no_grad():
    y_pred = model(x_test_tensor)
    y_pred = (y_pred > 0.5).float()
```

```
evaluator.confusion_matrix(y_test_tensor, y_pred)
```

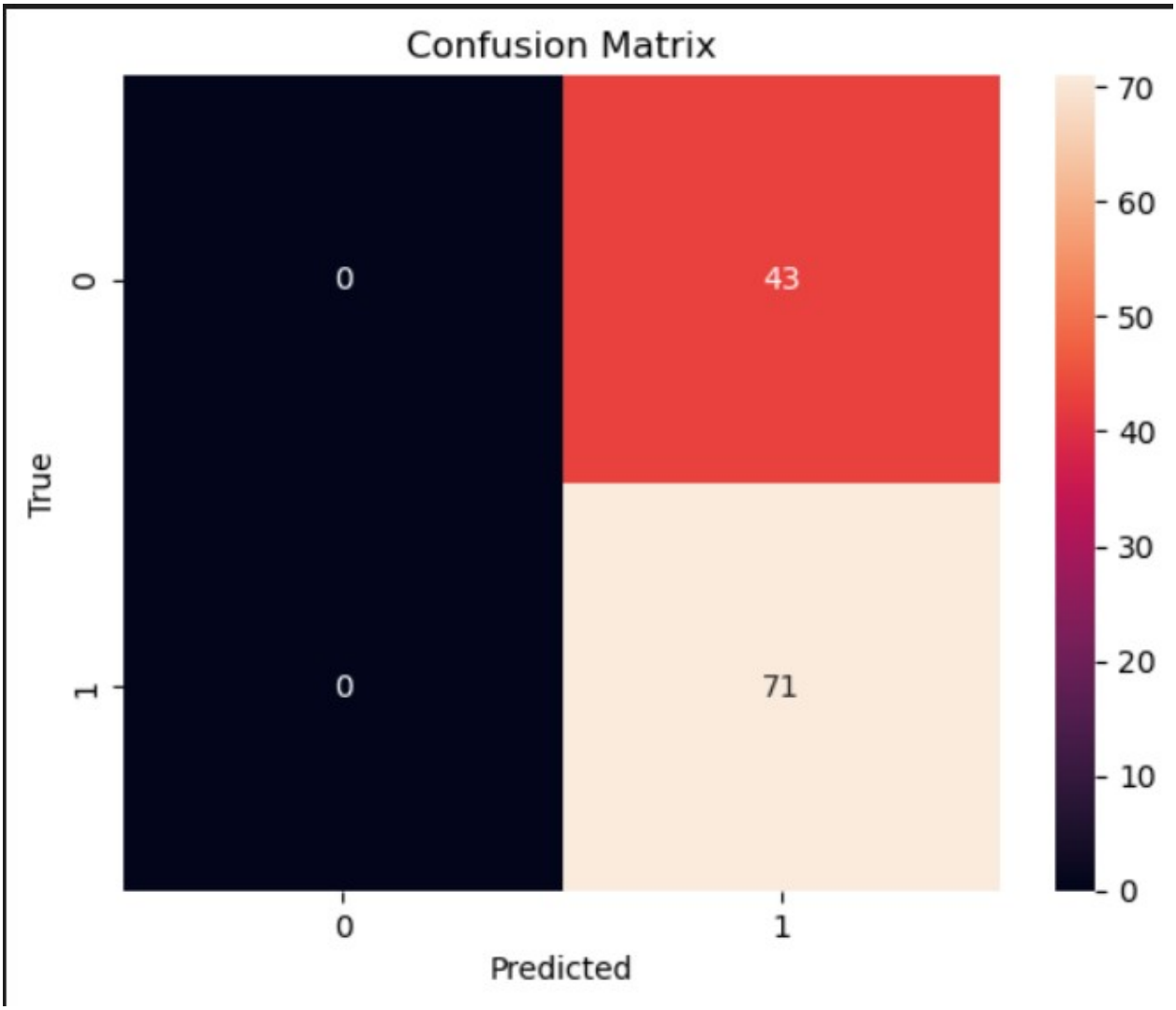
3.9 Link to github repository and colab applications :

Note : For more practice and to learn more, we can visit this tutorial.

Find the link to github repository

Link to Colab notebook

Link to Colab notebook

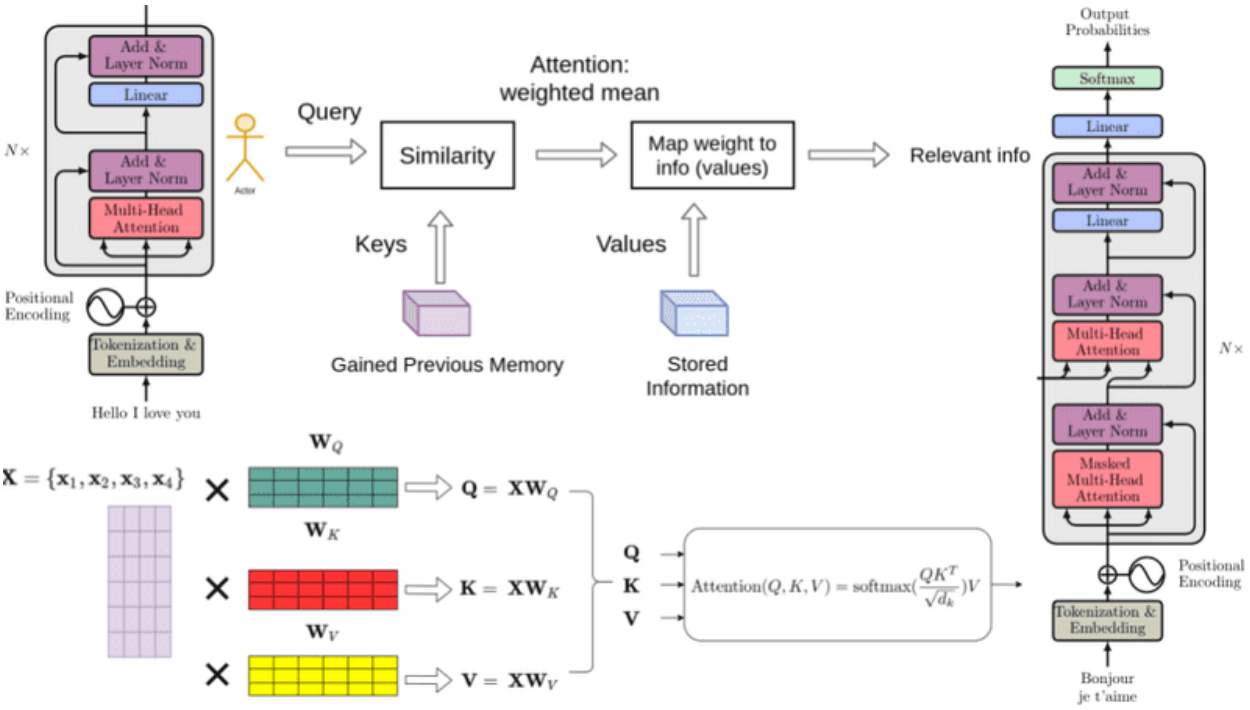

Confusion Matrix
0
43
0
71
True
0
1
0
1
Predicted
70
60
50
40
30
20
10
0

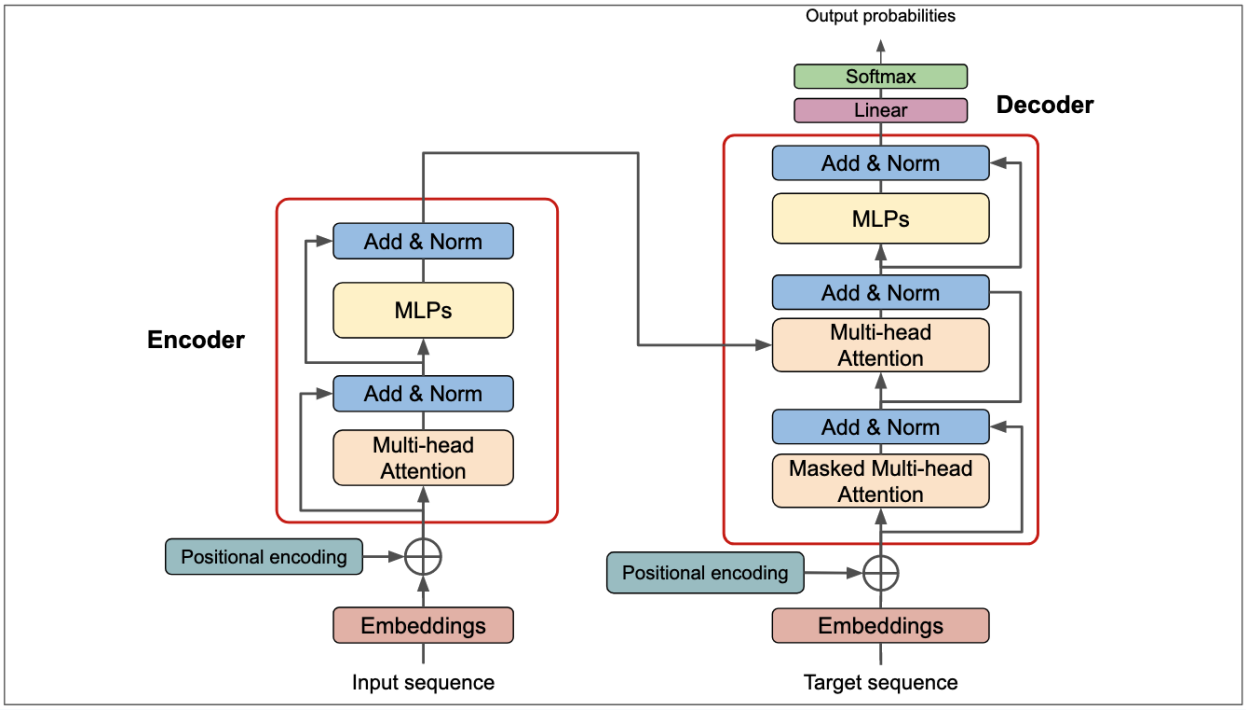
CHAPITRE 4

Transformer Architecture

4.1 1. General Introduction

4.1.1 a. *Overview*





4.1.2 b. *The purpose of Transformer networks*

4.1.3 c. *The Transformer Architecture*

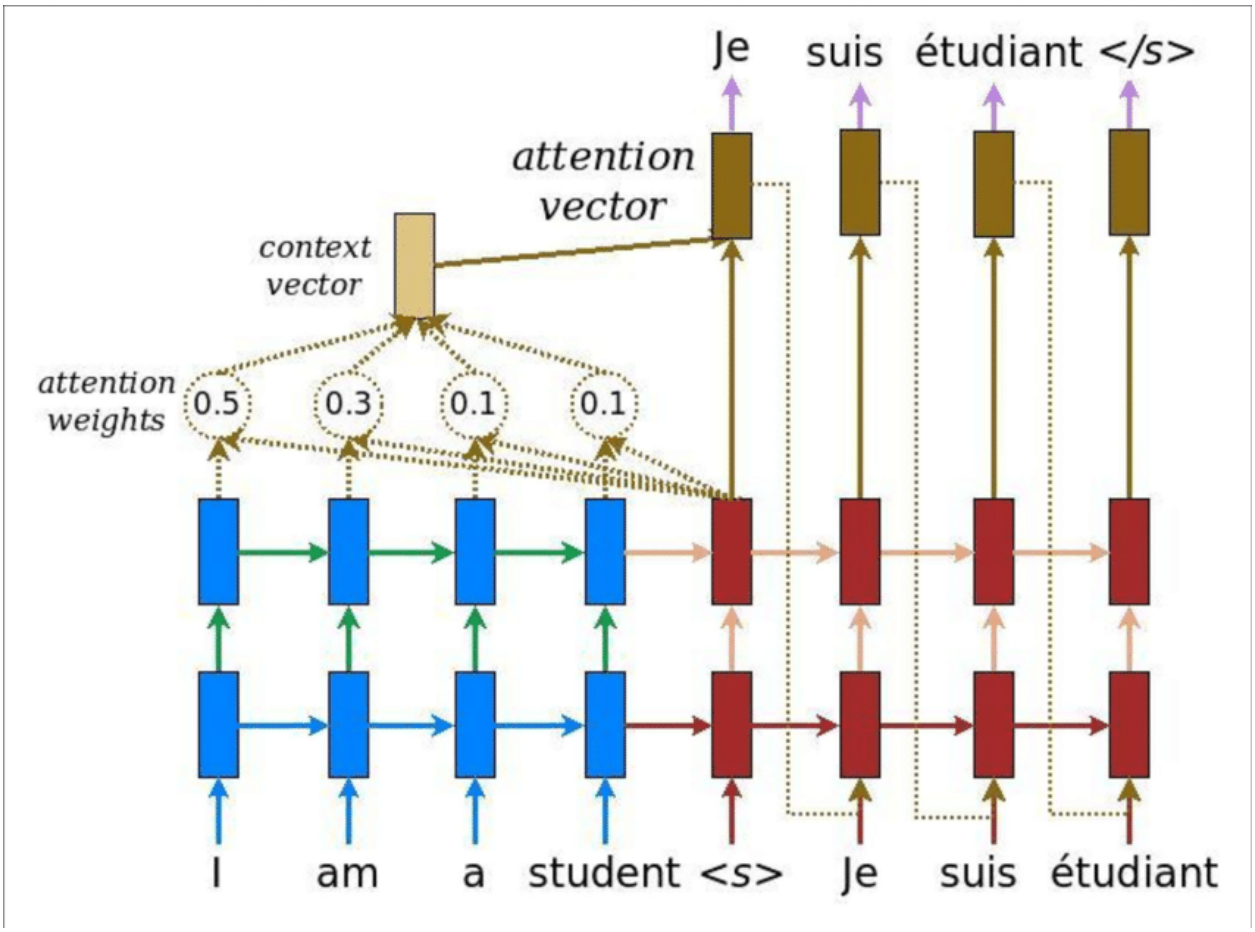
Note : At each step the model is auto-regressive, consuming the previously generated symbols as additional input when generating the next.

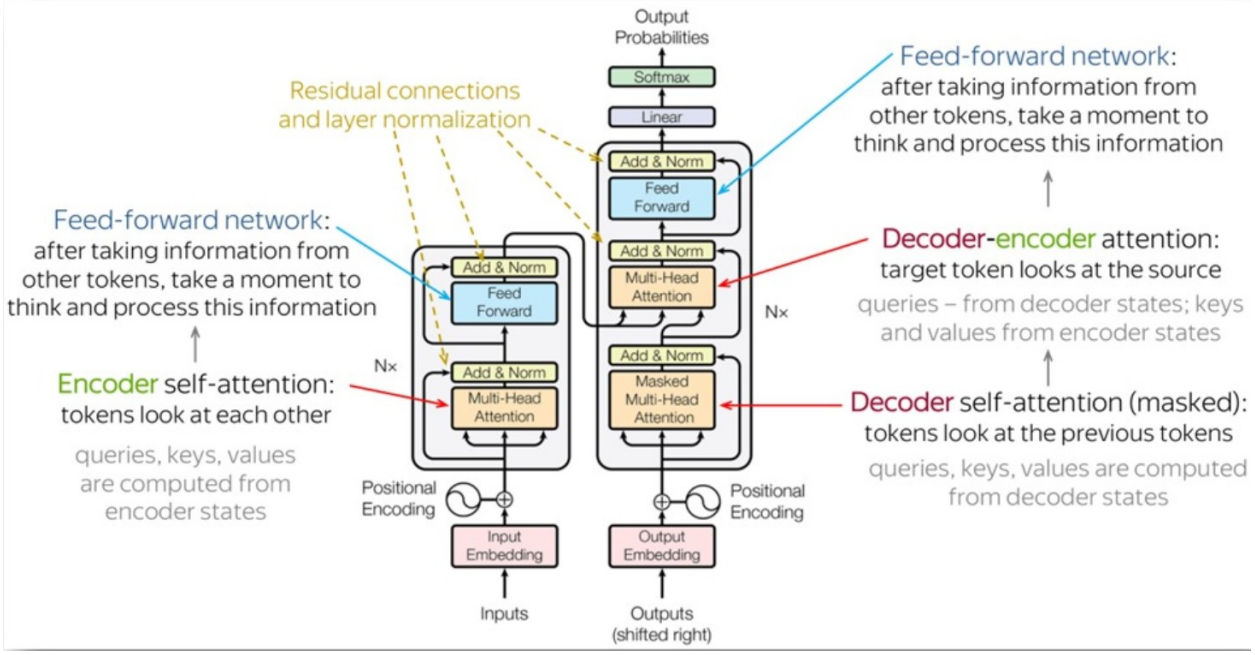
4.1.4 d. *Key Components*

4.2 2. The Encoder

4.2.1 a. *Tokenizer*

Note : More details in Tokenization in Machine Learning Explained


Je
suis
étudiant </s>
attention vector
context vector
attention weights
0.5
0.3
0.1
0.1
I
am
a
student <s>
Je
suis
étudiant


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)
Residual connections and layer normalization
Feed-forward network: after taking information from other tokens, take a moment to think and process this information
Feed-forward network: after taking information from other tokens, take a moment to think and process this information
Encoder self-attention: tokens look at each other
queries, keys, values are computed from encoder states
Decoder-encoder attention: target token looks at the source
queries – from decoder states; keys and values from encoder states
Decoder self-attention (masked): tokens look at the previous tokens
queries, keys, values are computed from decoder states

Token	ID
Hello	1
Go	2
ed	3
..	..
red	32000

4.2.2 b. *Input embedding*

Note : More details in Transformer Positional Embeddings and Encodings

4.2.3 c. *Positional Encoding*

Note : More details in Transformer Positional Embeddings and Encodings

4.2.4 d. *self Attention*

Note : self-attention (sometimes KQV-attention) layer is central mechanism in transformer architecture introduced in **`Attention Is All You Need paper<https ://arxiv.org/pdf/1706.03762.pdf>`__**

Note : More details in paper Attention is all you need : dot-product is "scaled", residual connection, layer normalization

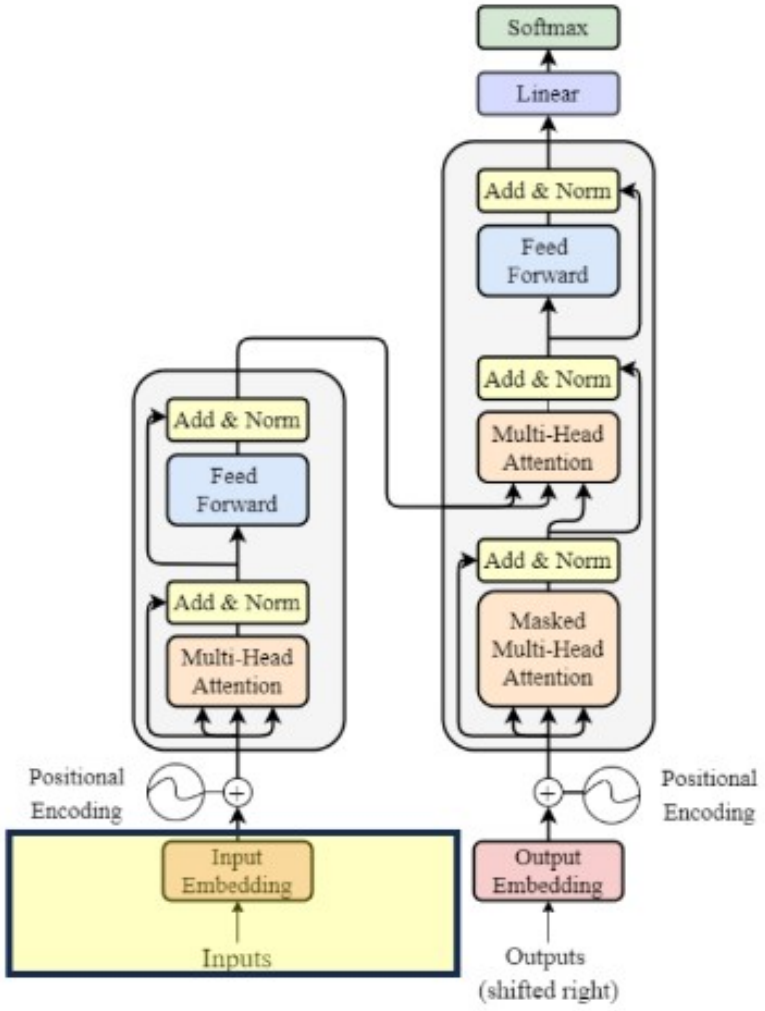
4.2.5 e. *Multi-Head Attention*

4.2.6 f. Add and norm - Norm

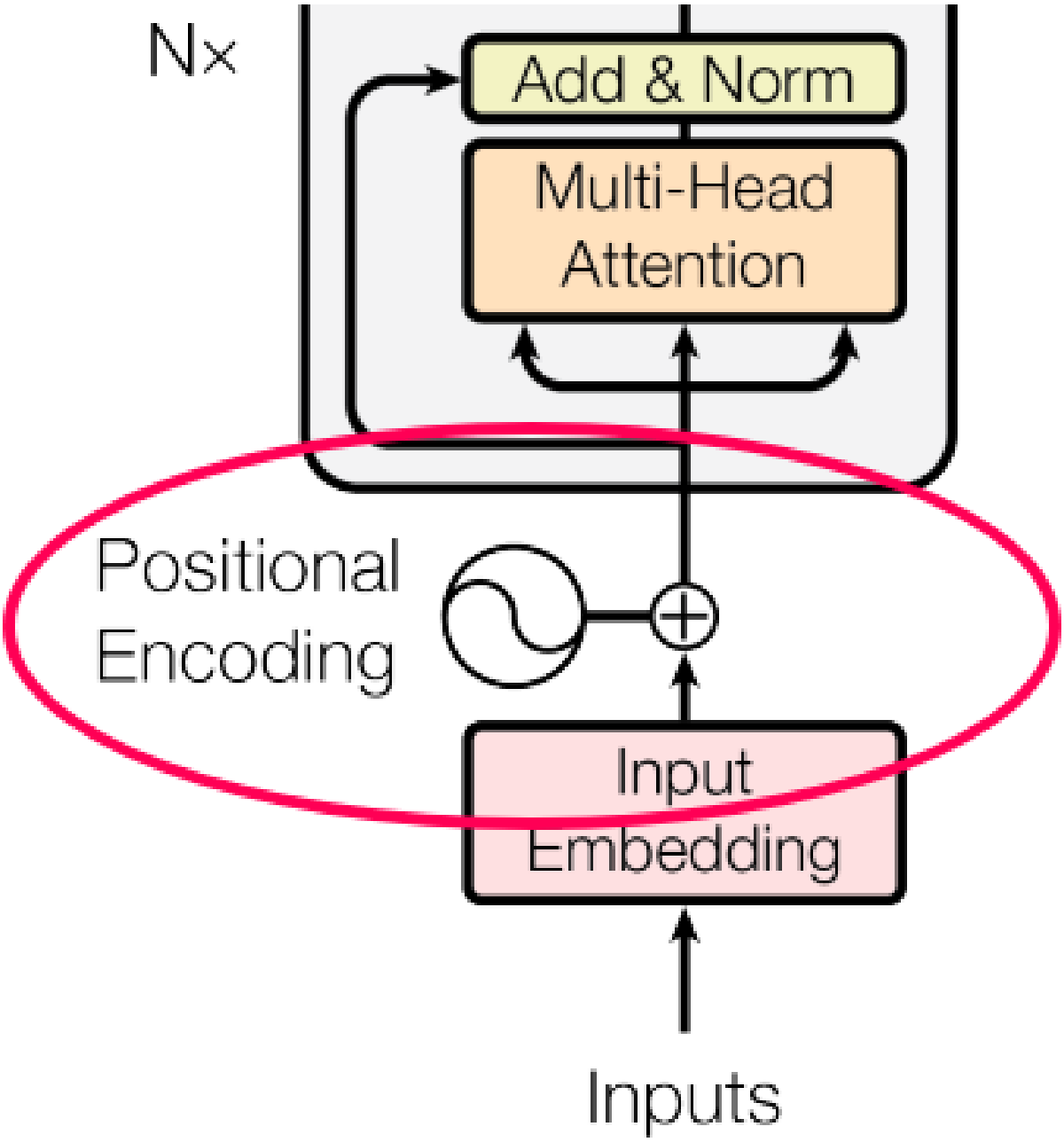
4.2.7 g. Feed Forward

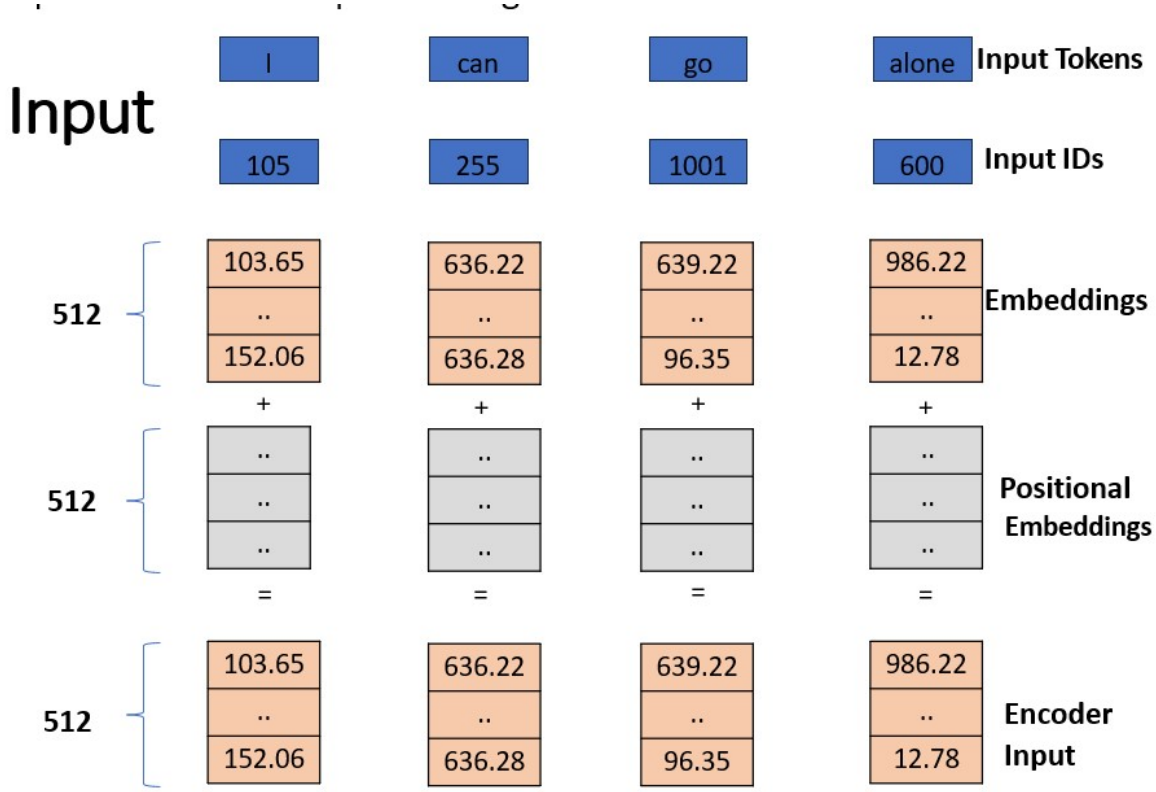
Mathematical Intuition

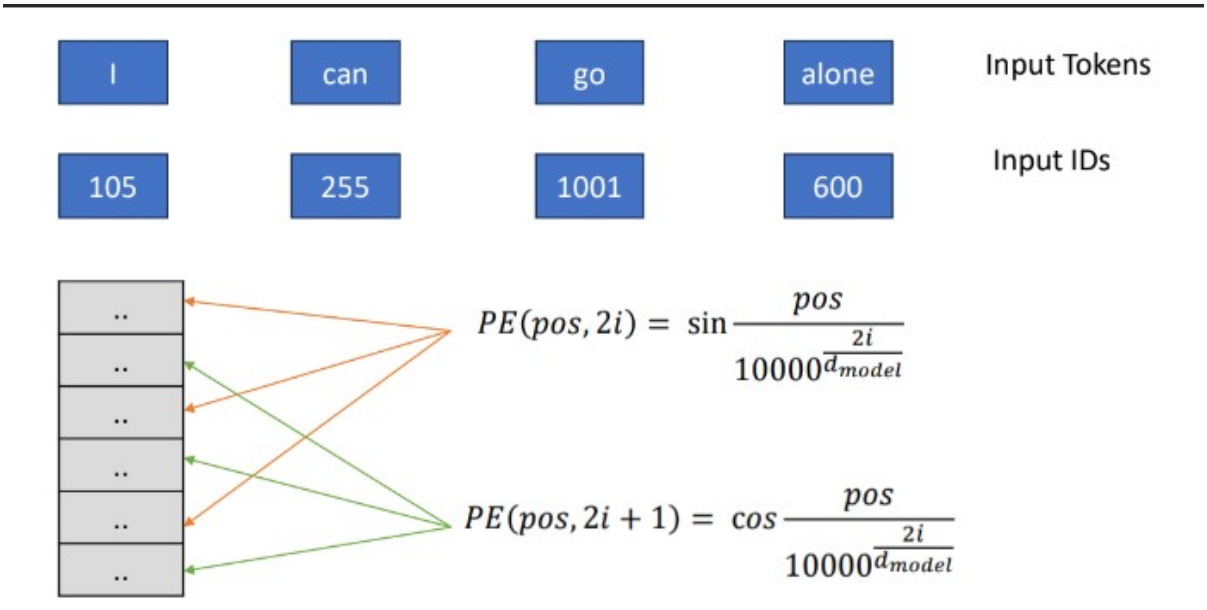
Note :


Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)


I can go alone
I
can
go
alone
Input Tokens
105
255
1001
600
Input IDs
Embeddings
512
103.65
633.01
25.33
..
152.06
636.22
2.01
96.25
..
636.28
639.22
9.36
78.22
..
96.35
986.22
7.22
9.36
..
12.78


Nx
Add & Norm
Multi-Head Attention
Positional Encoding
Input Embedding
Inputs


Input
I
can
go
alone
Input Tokens
105
255
1001
600
Input IDs
512
103.65
..
152.06
636.22
..
636.28
639.22
..
96.35
986.22
..
12.78
Embeddings
+
512
Positional
Embeddings
=
512
Encoder
Input


I
can
go
alone
Input Tokens
105
255
1001
600
Input IDs
$PE(pos, 2i) = \sin \frac{pos}{10000^{\frac{2i}{d_{model}}}}$
$PE(pos, 2i+1) = \cos \frac{pos}{10000^{\frac{2i}{d_{model}}}}$

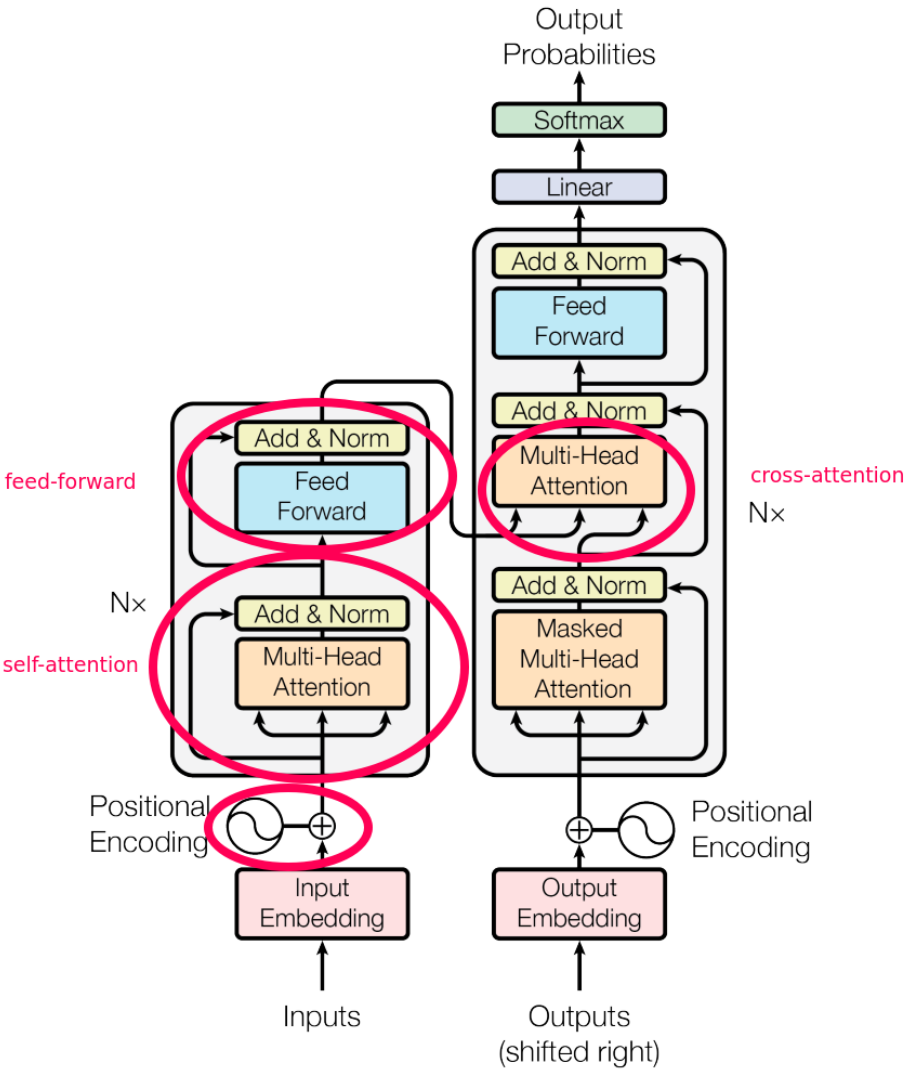
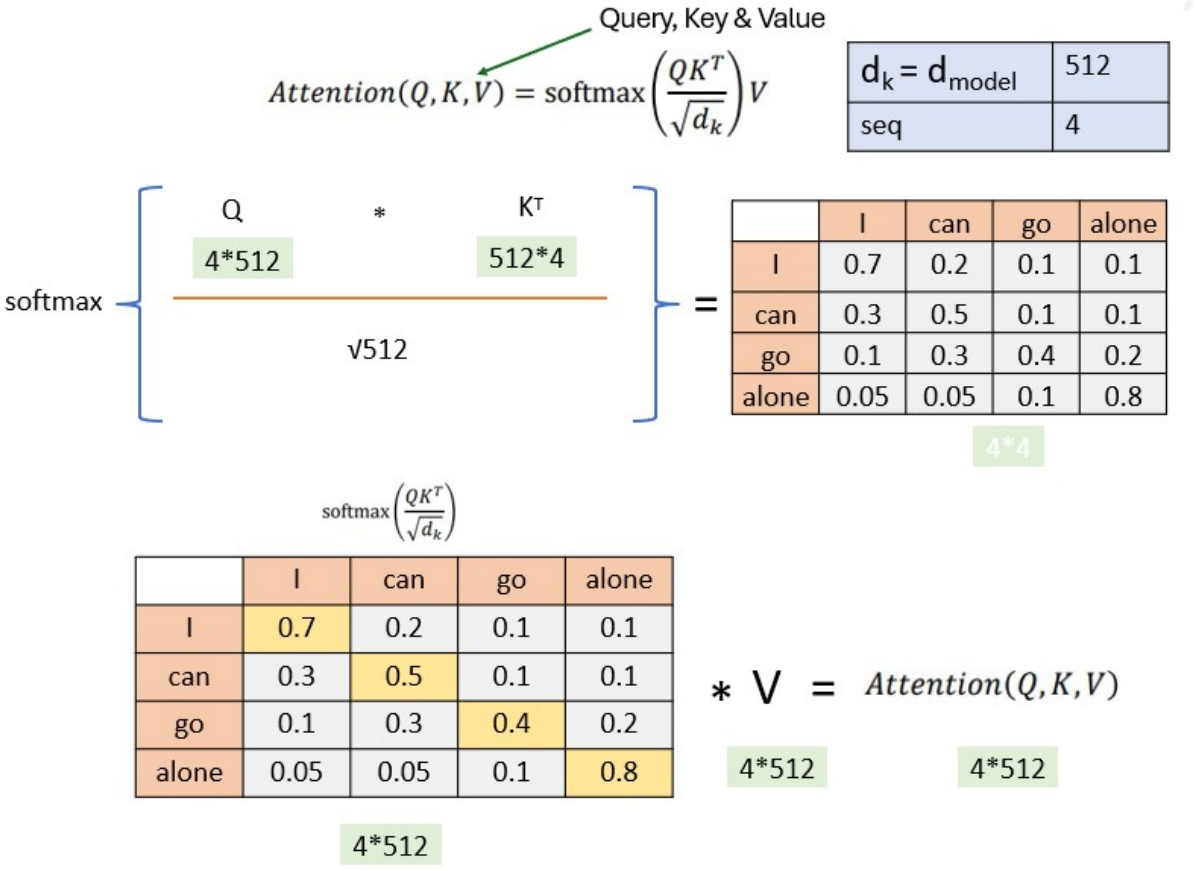
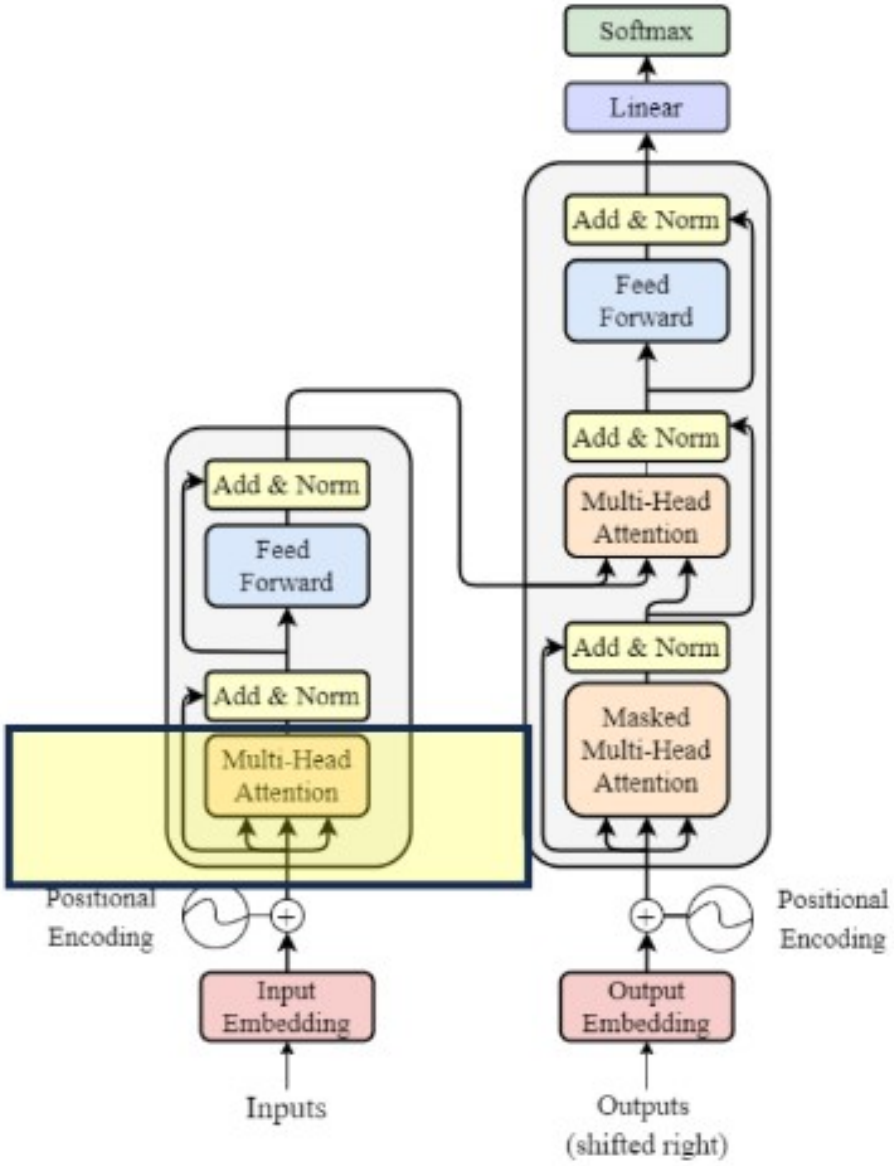


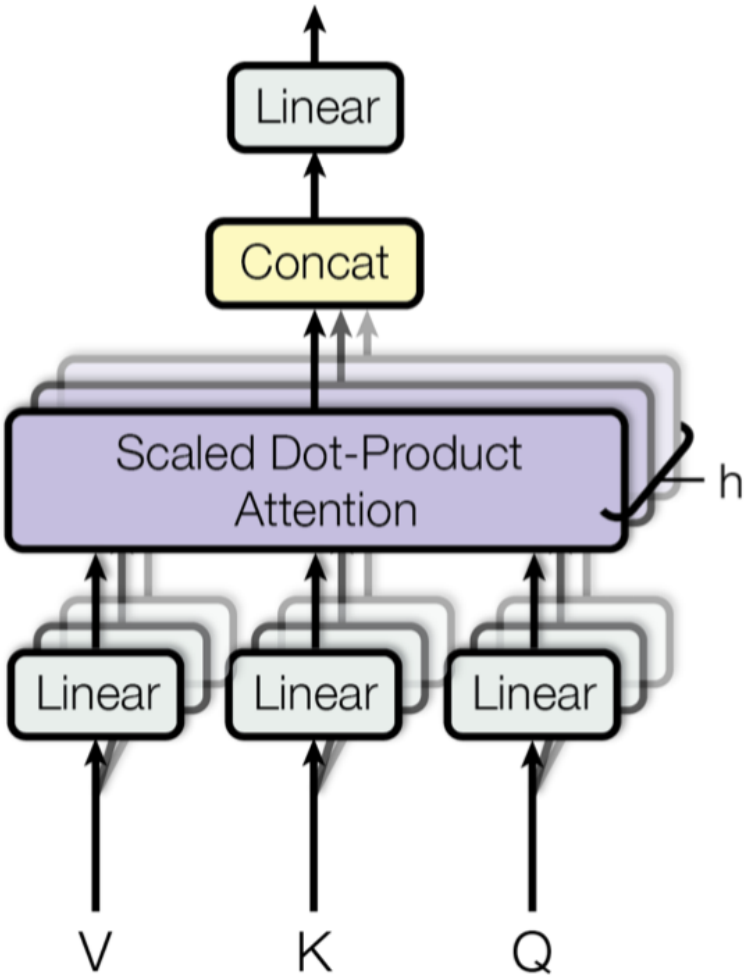
Figure 1: The Transformer - model architecture.

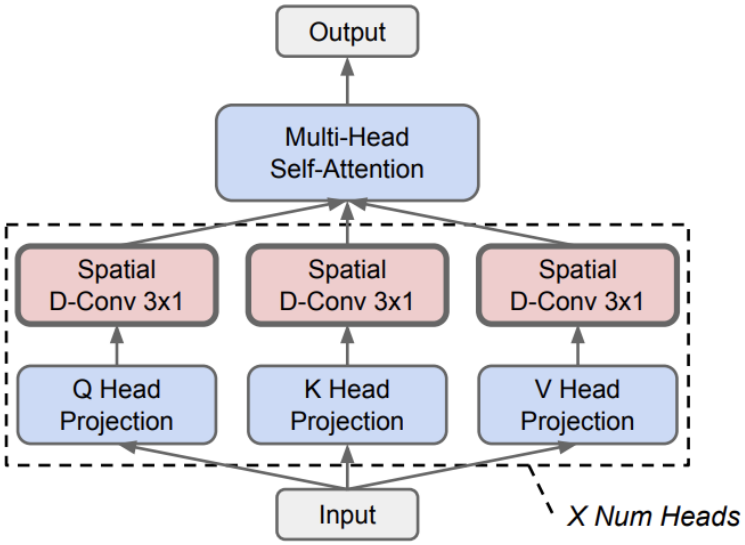

Query, Key & Value
$Attention(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$
$d_k = d_{model}$ 512
seq 4
softmax
Q
4*512
*
K^T
512*4
√512
=
I can go alone
I 0.7 0.2 0.1 0.1
can 0.3 0.5 0.1 0.1
go 0.1 0.3 0.4 0.2
alone 0.05 0.05 0.1 0.8
4*4
$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$
I can go alone
I 0.7 0.2 0.1 0.1
can 0.3 0.5 0.1 0.1
go 0.1 0.3 0.4 0.2
alone 0.05 0.05 0.1 0.8
4*512
* V = $Attention(Q,K,V)$
4*512
4*512

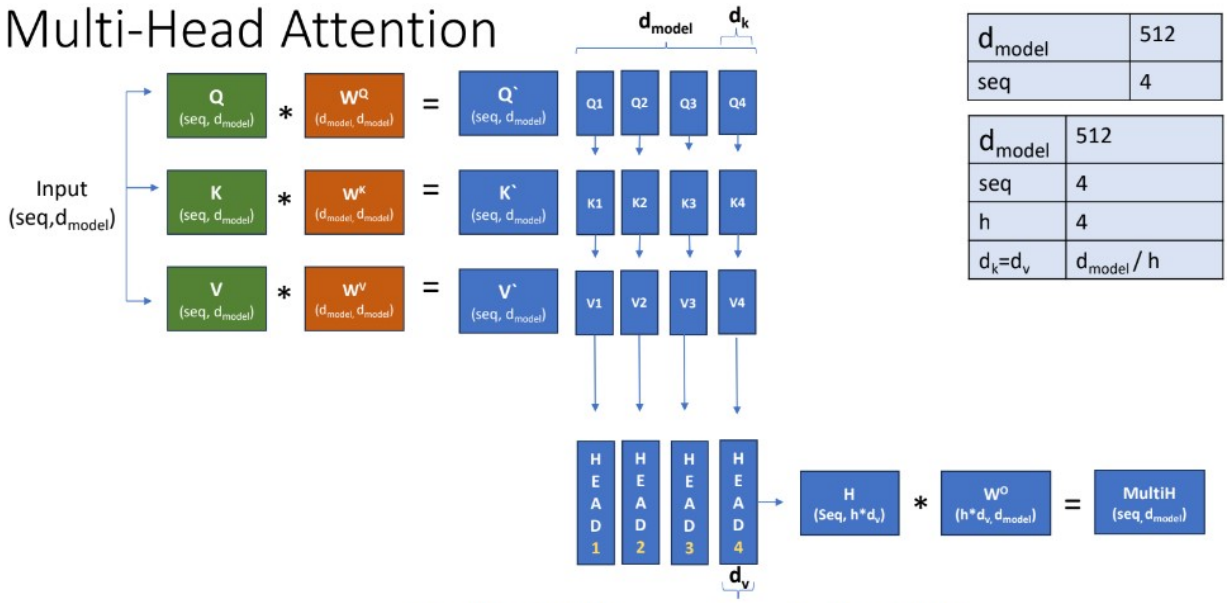


$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = [\text{head}_1, \ldots, \text{head}_h]\mathbf{W}_0$$

$$\text{where head}_i = \text{Attention}\left(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V\right)$$


Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q

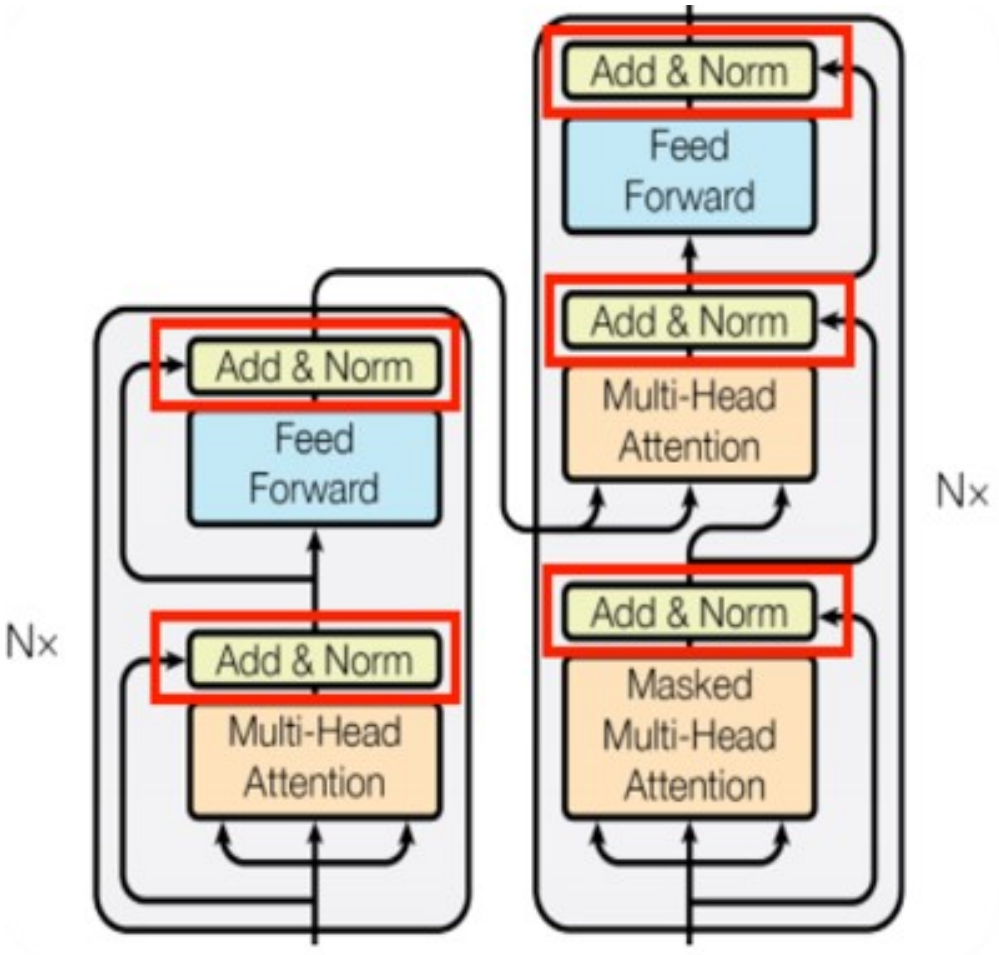

Output
Multi-Head Self-Attention
Spatial D-Conv 3x1
Spatial D-Conv 3x1
Spatial D-Conv 3x1
Q Head Projection
K Head Projection
V Head Projection
Input
X Num Heads

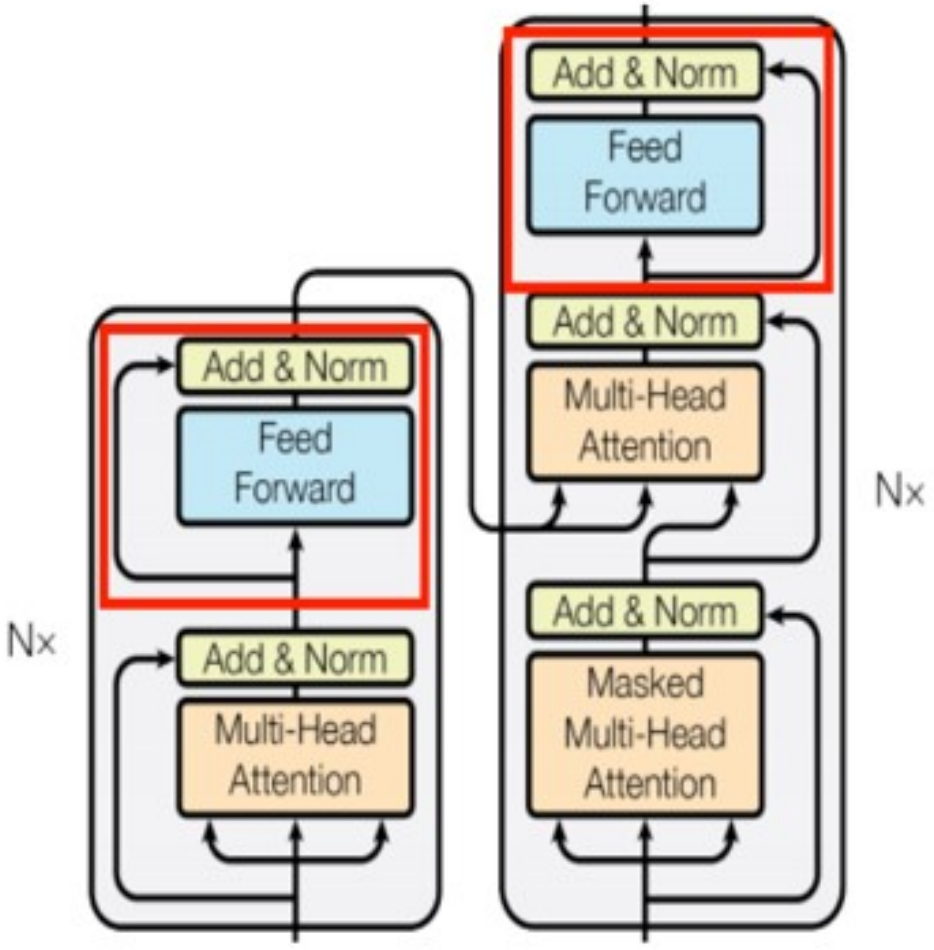


d_{model}	512
seq	4

d_{model}	512
seq	4
h	4
$d_k=d_v$	d_{model} / h

$$MultiHead(Q,K,V) = Concat(head_1 \ ... head_h)W^O$$





$$\mathbf{y}^{\ell} = \sum_{i} \mathrm{ReLU}(\mathbf{x}^{\ell} \cdot \mathbf{k}_{i}^{\ell}) \cdot \mathbf{v}_{i}^{\ell} + \mathbf{b}^{\ell}.$$

4.2.8 h. *Residual Connections*

Note :

4.2.9 i. *Conclusion*

Note :

— You can view more by clicking the « Transformer's Encoder »

4.2.10 j. *BIBLIOGRAPHIC*

For more information

— « what is a transformer »
— « Transformers-Encoder-Decoder » vidéo YouTube
— « transformers-important-architecture »
— « transformers-introduction »
— « Attention is all you need » vidéo YouTube
— Mécanismes d'attention

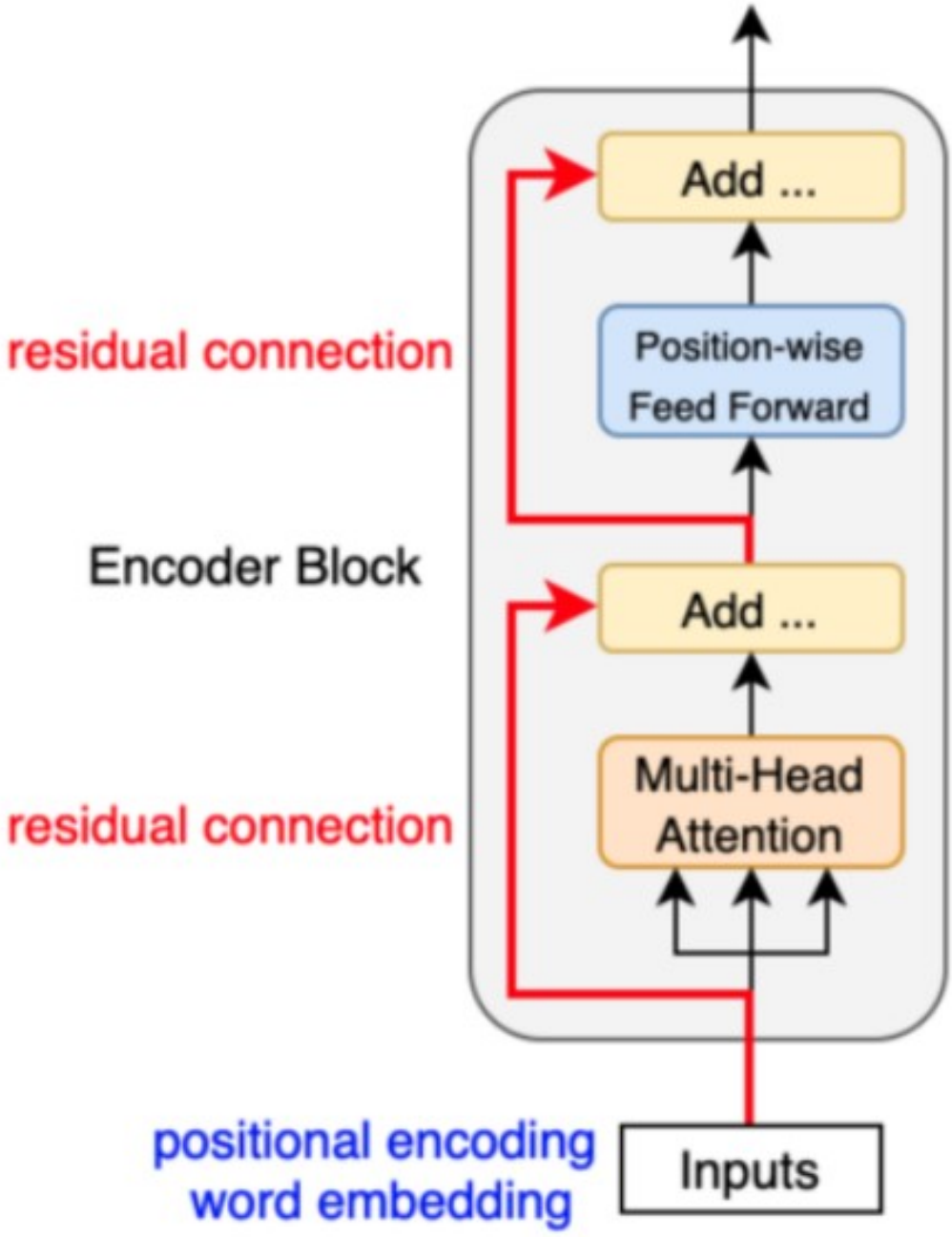
4.3 3. The Decoder

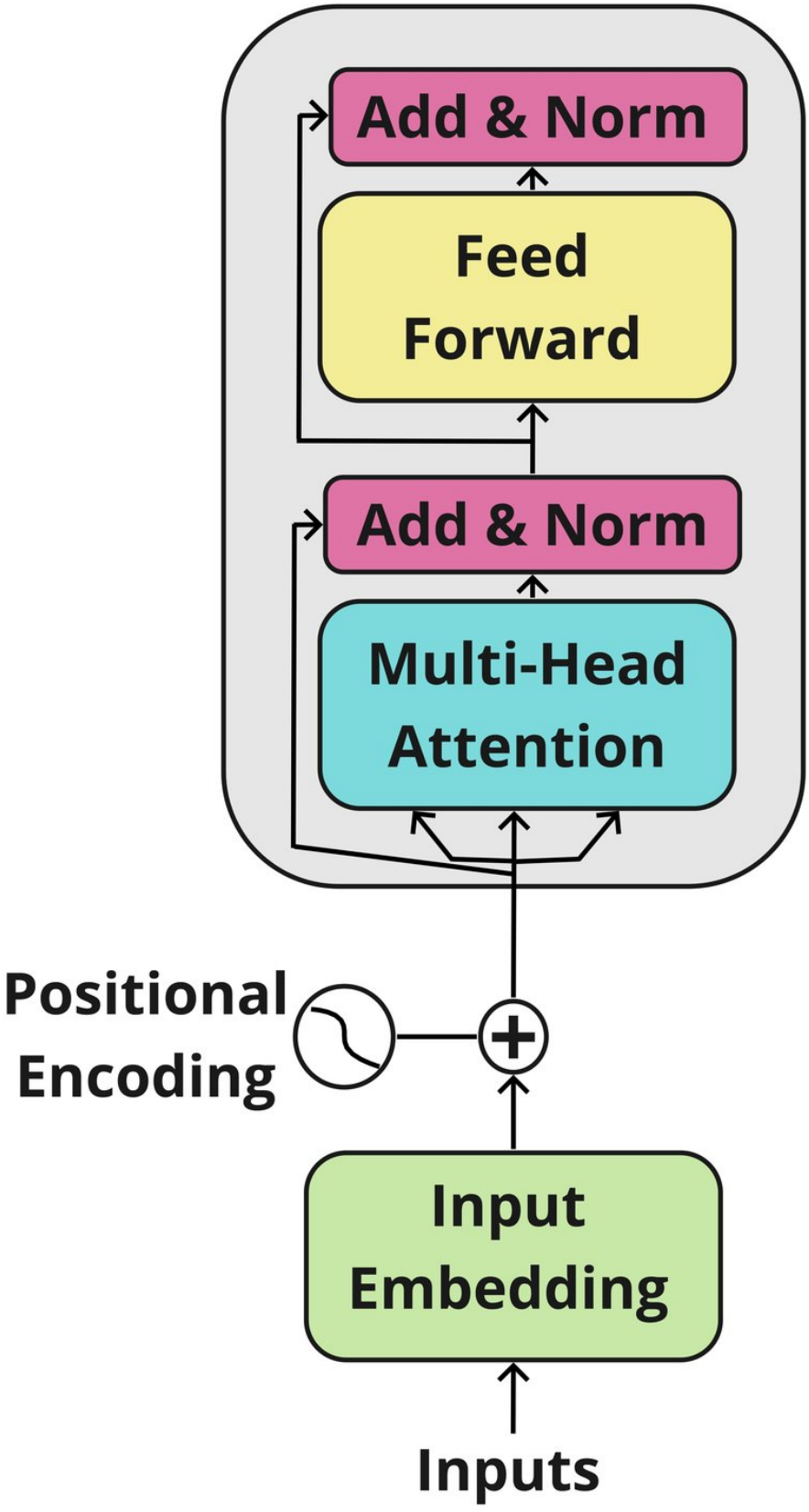
Note : The decoder block is similar to the encoder block, except it calculates the source-target attention.

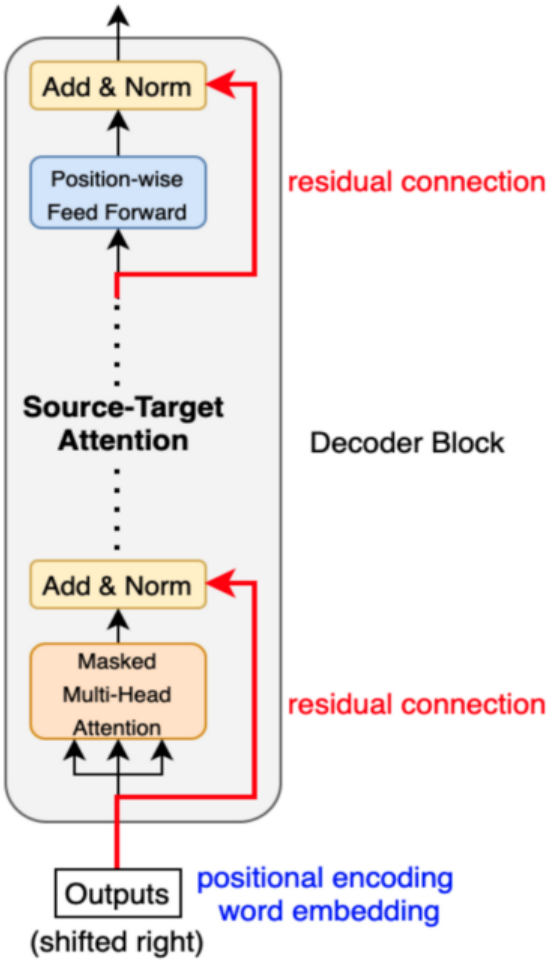
Overview

4.3.1 a. *Introduction*

Note : The decoder and encoder share several similarities, but they differ in their input. We will explain this difference in the next section.


residual connection
Encoder Block
residual connection
positional encoding
word embedding
Add ...
Position-wise
Feed Forward
Add ...
Multi-Head
Attention
Inputs


Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Positional Encoding
Input Embedding
Inputs


Add & Norm
Position-wise
Feed Forward
residual connection
Source-Target
Attention
Decoder Block
Add & Norm
Masked
Multi-Head
Attention
residual connection
Outputs
(shifted right)
positional encoding
word embedding

4.3.2 b. *difference between decoder and encoder*

$$\text{mask}(QK^T) = \text{mask}\left(\begin{bmatrix} e_{11} & e_{12} & \dots & e_{1n} \\ e_{21} & e_{22} & \dots & e_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ e_{m1} & e_{m2} & \dots & e_{mn} \end{bmatrix}\right) = \begin{bmatrix} e_{11} & -\infty & \dots & -\infty \\ e_{21} & e_{22} & \dots & -\infty \\ \vdots & \vdots & \ddots & \vdots \\ e_{m1} & e_{m2} & \dots & e_{mn} \end{bmatrix}$$

4.3.3 c. *Masked multi-head attention*

Note :

4.3.4 d. *Multi-Head Attention*

4.3.5 e. *Feed Forward*

Note : urthermore, the three sublayers on the decoder side also have residual connections around them and are succeeded by a normalization layer.

Positional encodings are also added to the input embeddings of the decoder in the same manner as previously explained for the encoder.

4.3.6 f. *Conclusion*

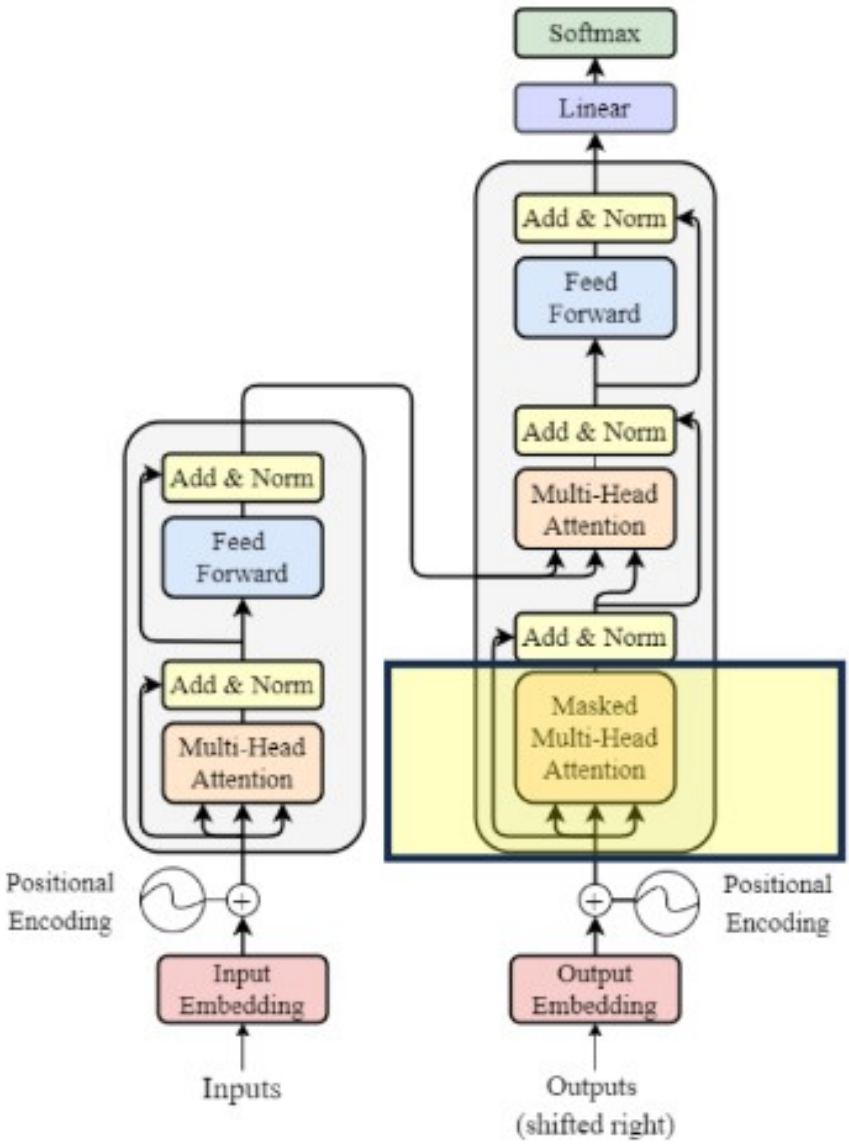
For more information

- « transformers-encoder-decoder »
- « the-transformer_decoder_model »
- « transformer-decoder »

4.4 3. Softmax in Transformers

$$\text{Softmax}(QK^\top) = \frac{\exp(QK_i^\top)}{\sum_j \exp(QK_j^\top)}$$

For more information



softmax(Q * K^T / √512) =

Q: 4*512, K^T: 512*4

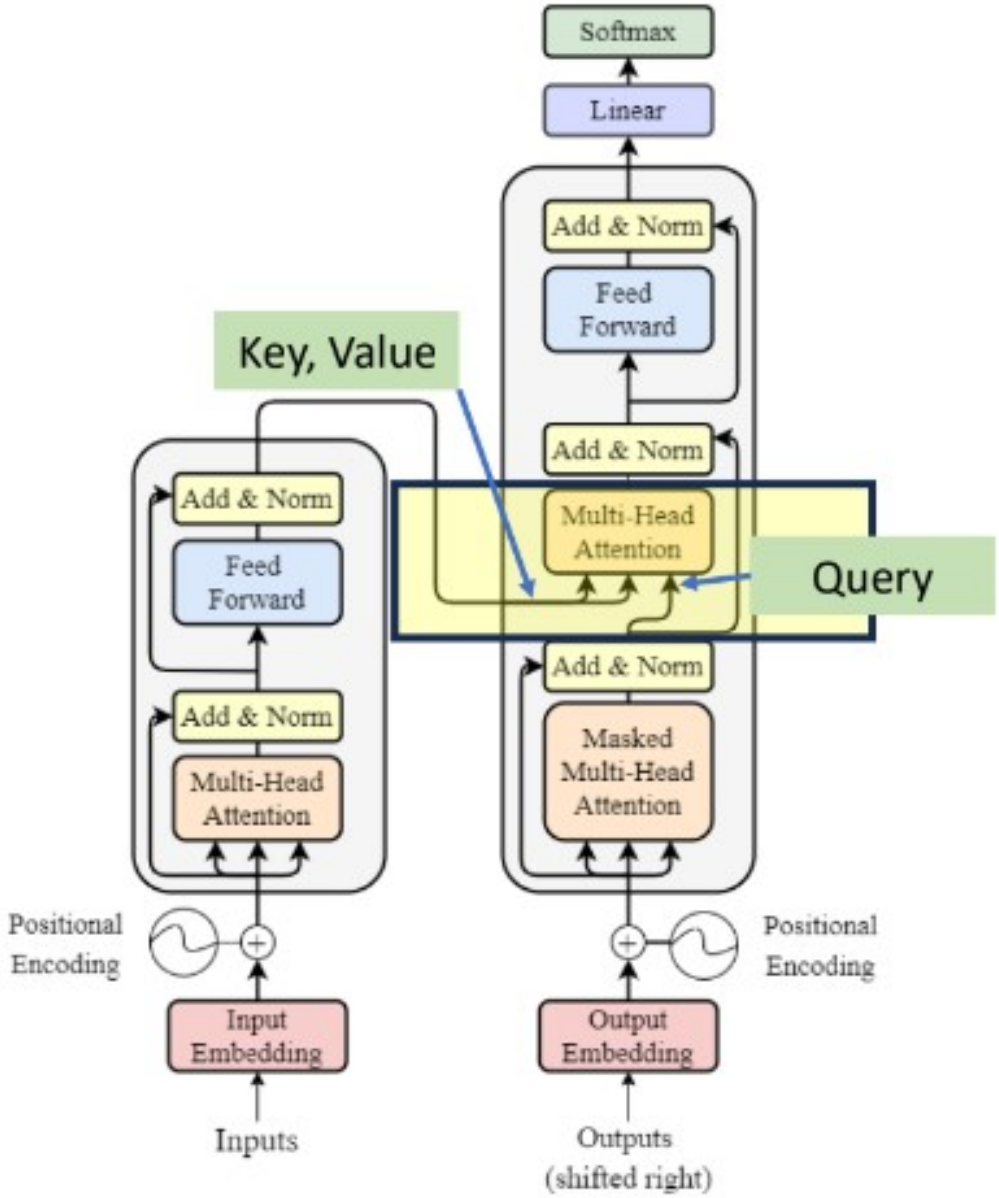
	I	can	go	alone
I	0.7	0.2	0.1	0.1
can	0.3	0.5	0.1	0.1
go	0.1	0.3	0.4	0.2
alone	0.05	0.05	0.1	0.8

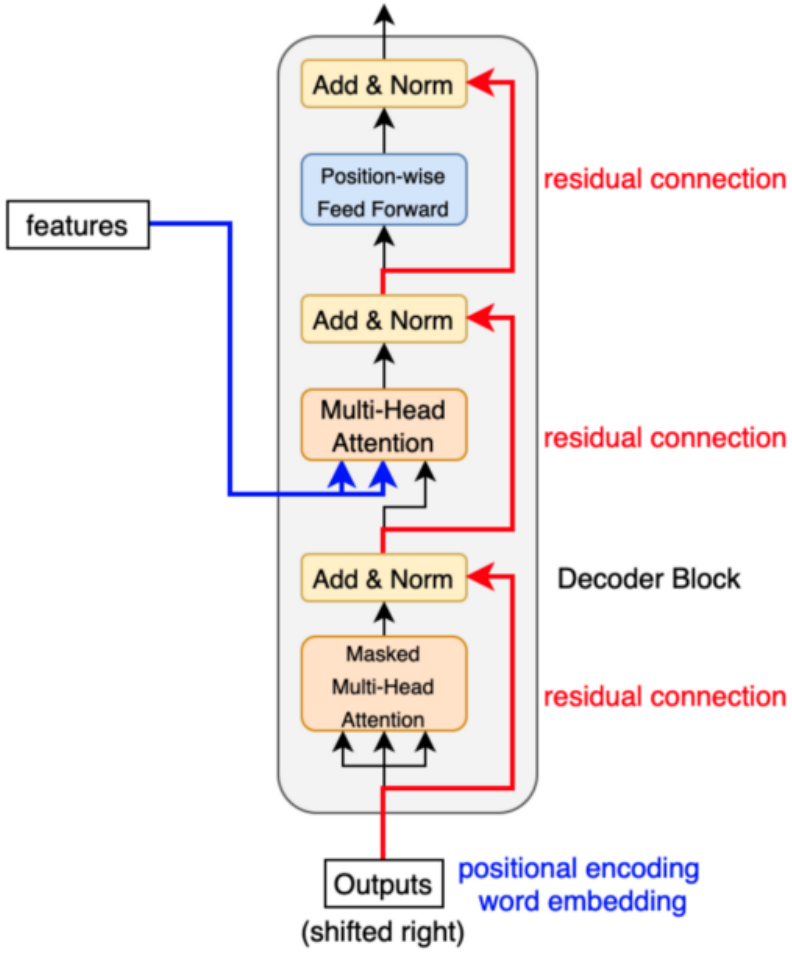
4*4

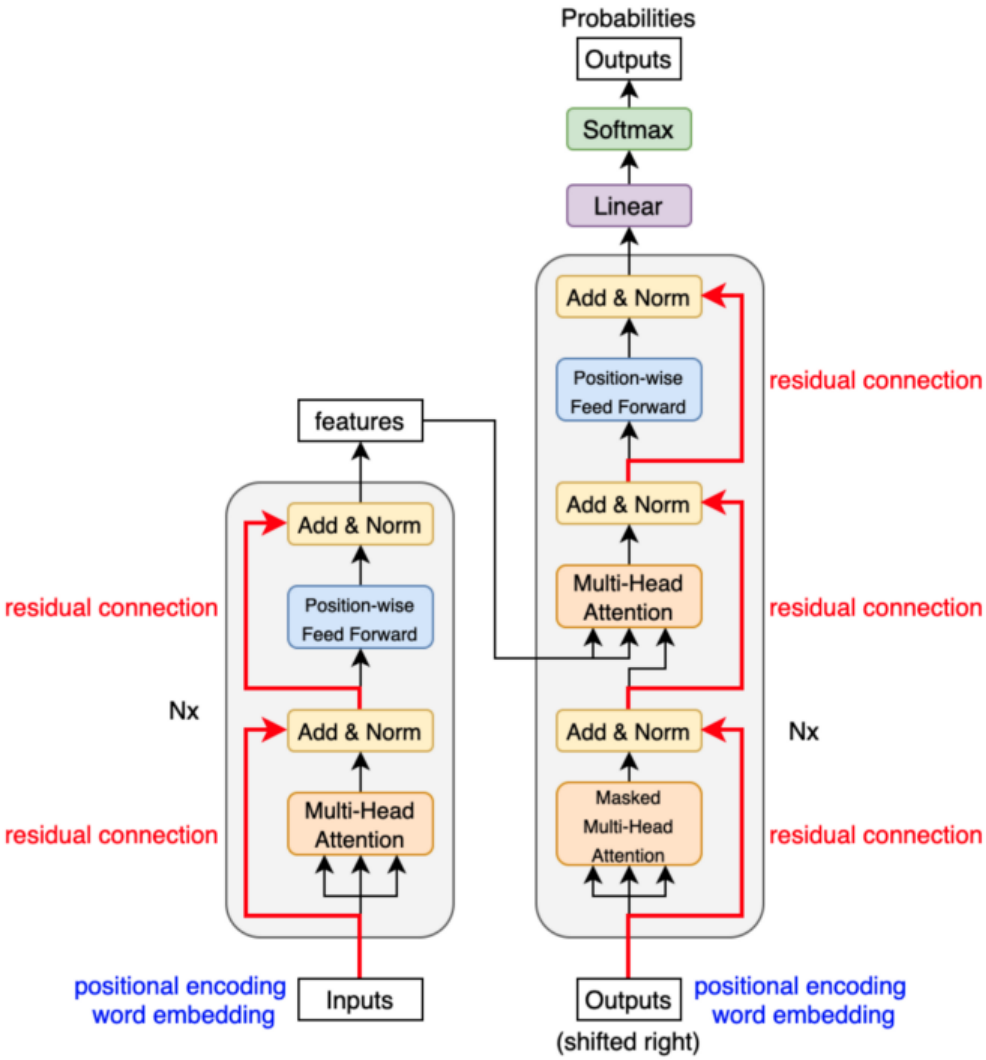
$\left(\frac{QK^T}{\sqrt{d_k}}\right)$

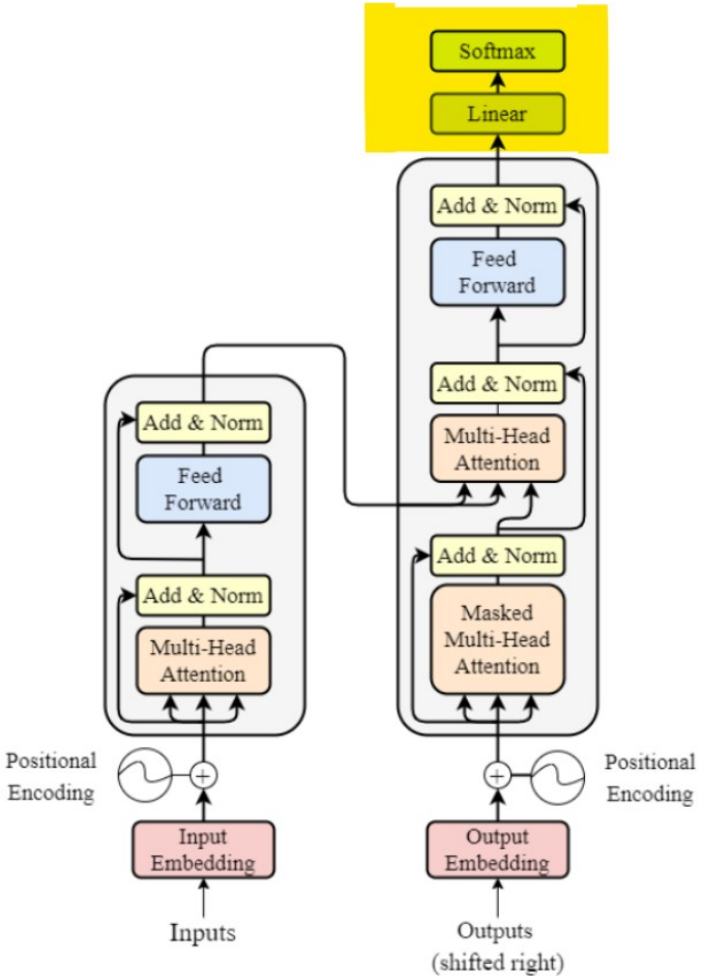
	I	can	go	alone
I	63.3	1.2	2.6	7.2
can	3.25	0.3	1.2	2.1
go	12.0	11.9	52.9	2.9
alone	1.6	63.1	14.2	101.3

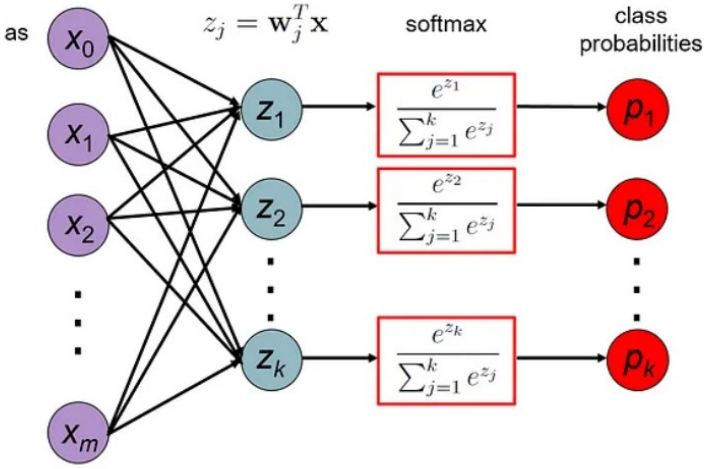
	I	can	go	alone
I	63.3	-∞	-∞	-∞
can	3.25	96.1	-∞	-∞
go	12.0	11.9	52.9	-∞
alone	1.6	63.1	14.2	101.3




Add & Norm
Position-wise
Feed Forward
residual connection
features
Add & Norm
Multi-Head
Attention
residual connection
Add & Norm
Decoder Block
Masked
Multi-Head
Attention
residual connection
Outputs
(shifted right)
positional encoding
word embedding


Probabilities
Outputs
Softmax
Linear
Add & Norm
Position-wise Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
residual connection
residual connection
residual connection
Nx
Nx
features
Add & Norm
Position-wise Feed Forward
Add & Norm
Multi-Head Attention
residual connection
residual connection
residual connection
positional encoding
word embedding
Inputs
Outputs
(shifted right)
positional encoding
word embedding


Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)


as
$z_j = \mathbf{w}_j^T \mathbf{x}$
softmax
class probabilities
x_0
x_1
x_2
x_m
z_1
z_2
z_k
$\frac{e^{z_1}}{\sum_{j=1}^{k} e^{z_j}}$
$\frac{e^{z_2}}{\sum_{j=1}^{k} e^{z_j}}$
$\frac{e^{z_k}}{\sum_{j=1}^{k} e^{z_j}}$
p_1
p_2
p_k

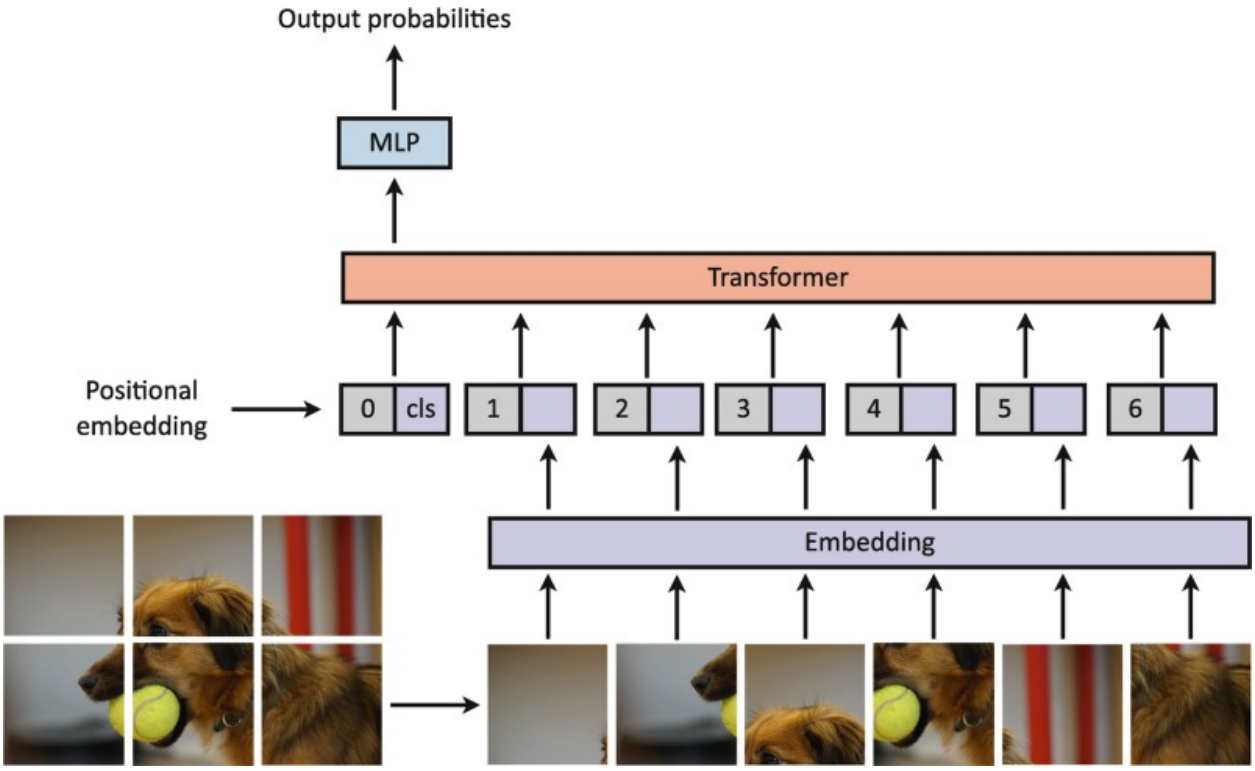
$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K.$$

— « why do we use softmax in transformers »
— « softmax paper link »

CHAPITRE 5

Visual Transformer (ViT)

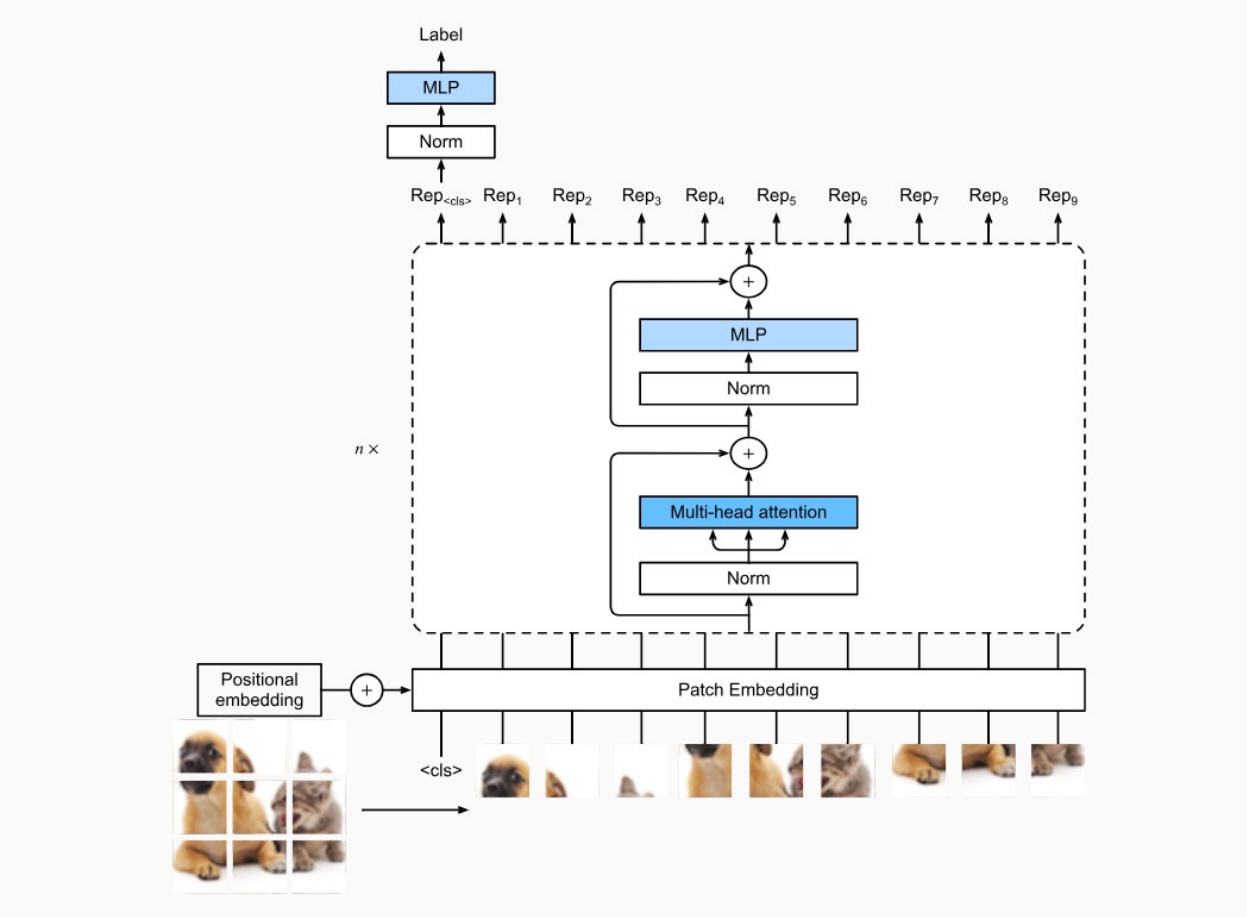
5.1 1. introduction



For more Understanding Vision Transformers

link for more information

— Vision Transformers



For more Understanding Vision Transformers

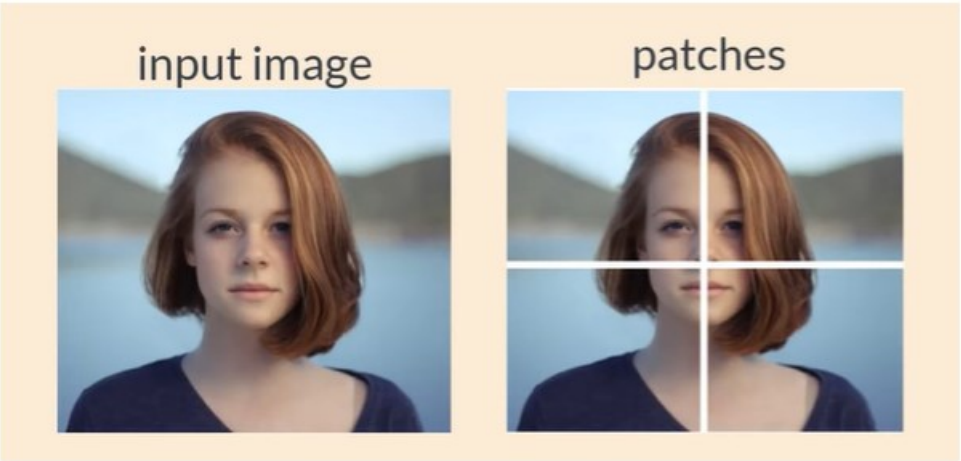
link for more information

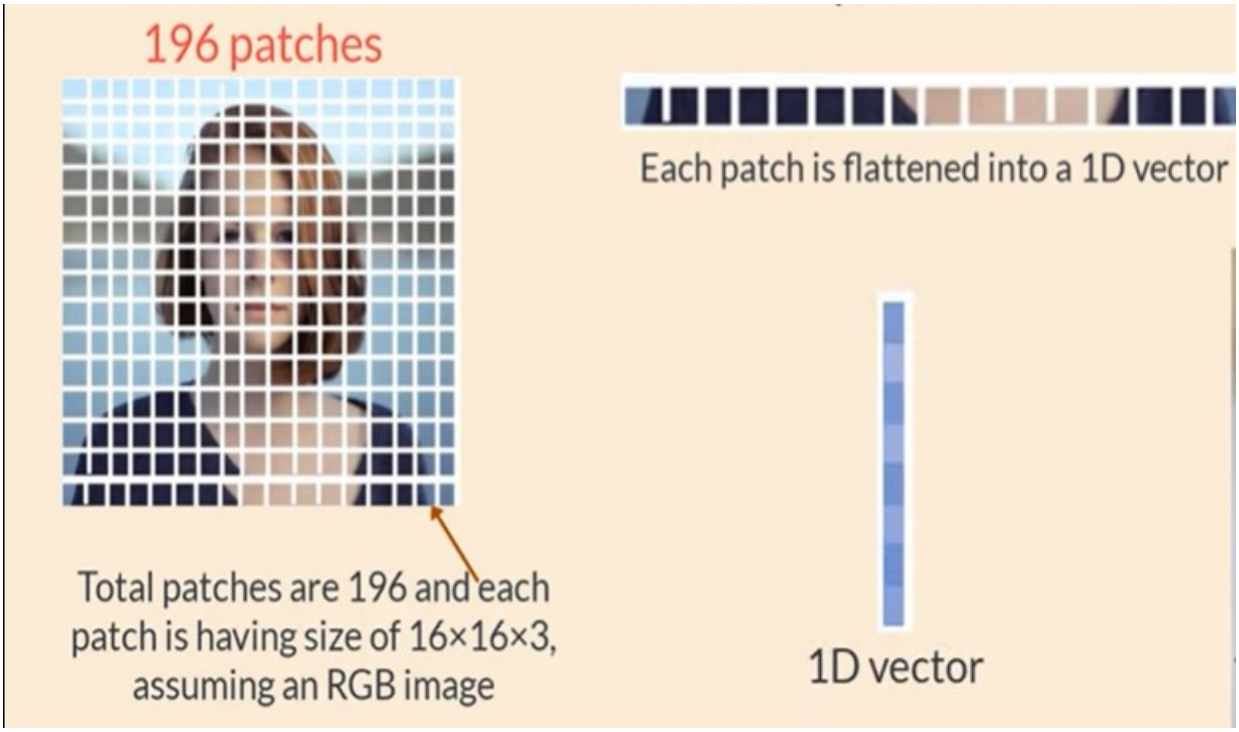
— The vision Transformer architecture

Note : Let's explain how the vision transformer works step by step with an example.

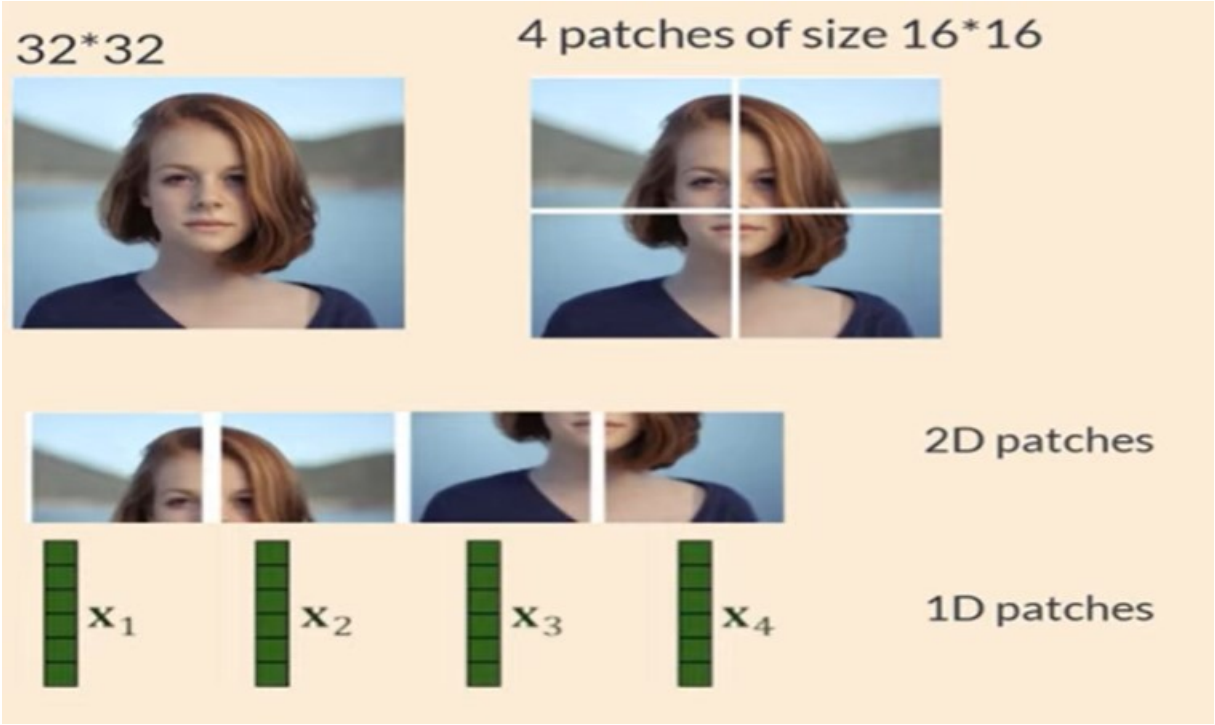
5.2 2. Vision Transformers example

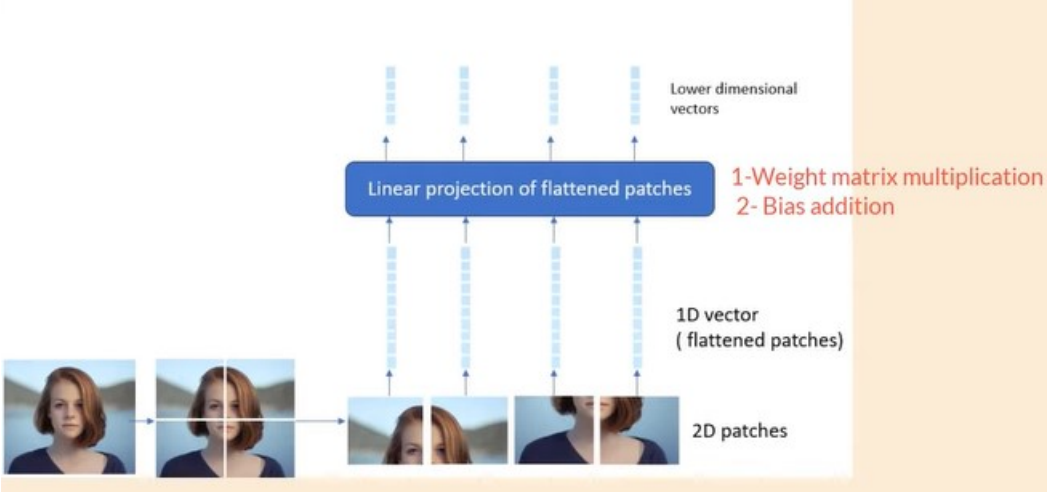
— **example**

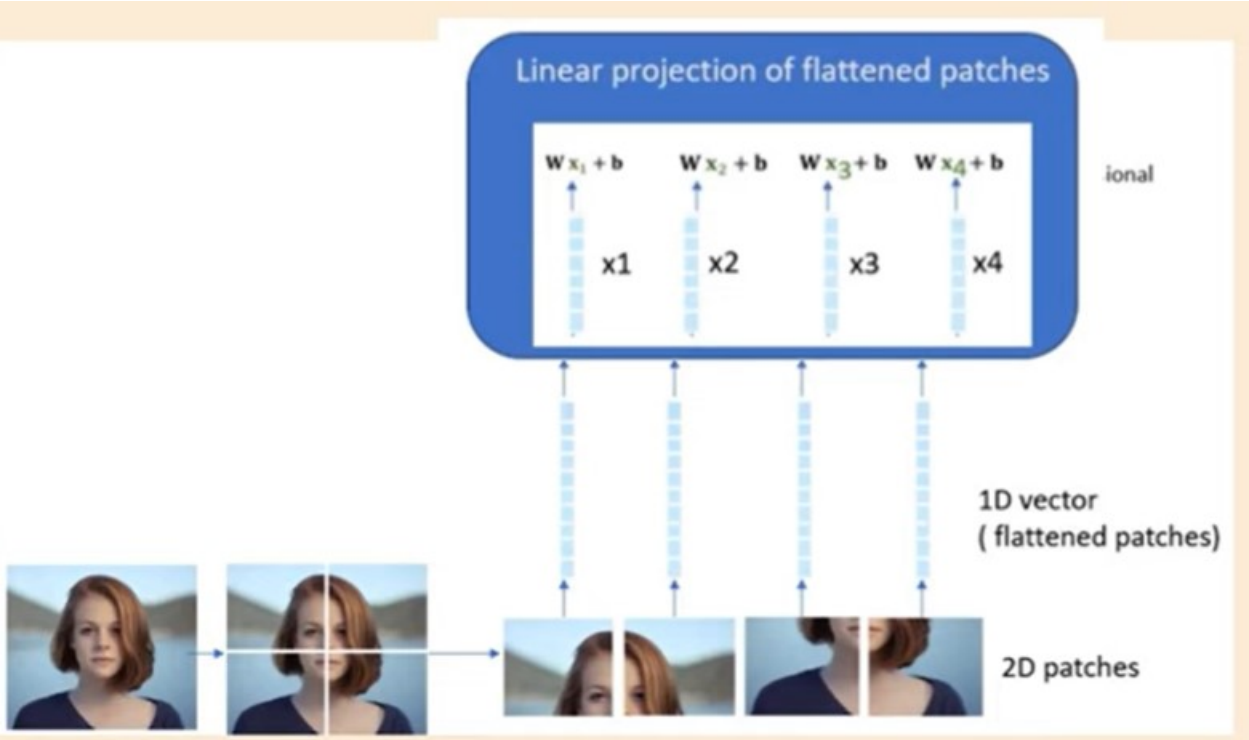


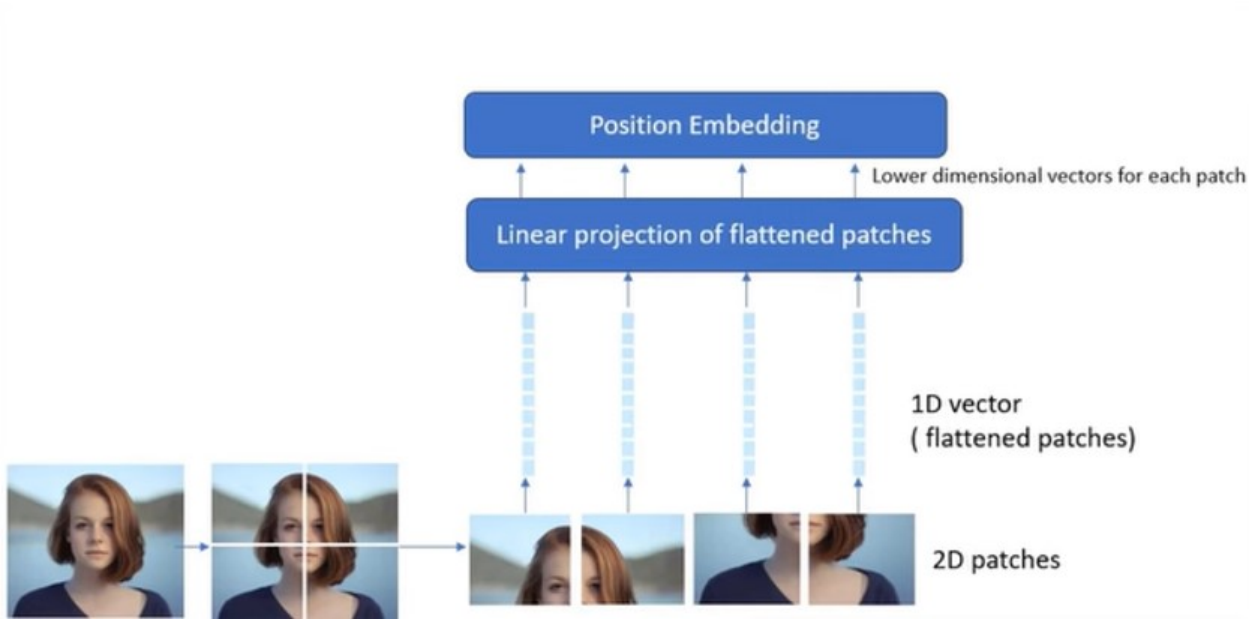

196 patches
Each patch is flattened into a 1D vector
Total patches are 196 and each patch is having size of 16×16×3, assuming an RGB image
1D vector

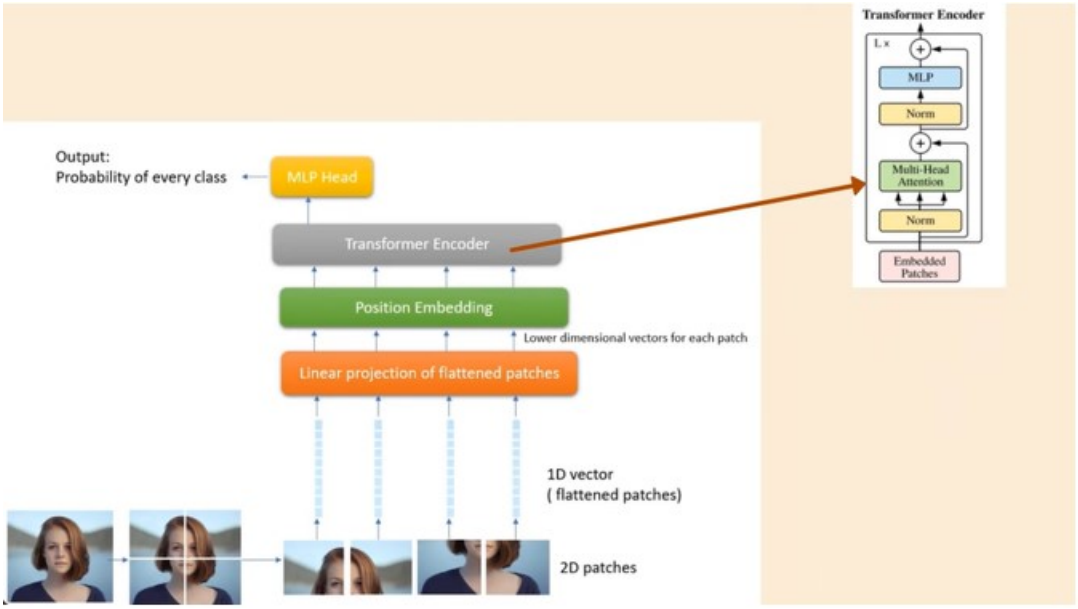

flattened patch
patch with a tree
16
16
patch = 16*16 =256 tokens


32*32
4 patches of size 16*16
2D patches
x_1
x_2
x_3
x_4
1D patches


Lower dimensional vectors
Linear projection of flattened patches
1-Weight matrix multiplication
2- Bias addition
1D vector
(flattened patches)
2D patches


Linear projection of flattened patches
$Wx_1 + b$
$Wx_2 + b$
$Wx_3 + b$
$Wx_4 + b$
x1
x2
x3
x4
ional
1D vector
(flattened patches)
2D patches


Position Embedding
Lower dimensional vectors for each patch
Linear projection of flattened patches
1D vector
(flattened patches)
2D patches


Transformer Encoder
L x
MLP
Norm
Multi-Head Attention
Norm
Embedded Patches
Output:
Probability of every class
MLP Head
Transformer Encoder
Position Embedding
Lower dimensional vectors for each patch
Linear projection of flattened patches
1D vector
(flattened patches)
2D patches

CHAPITRE 6

ViT project implemention

6.1 Table of Contents

6.2 1. Vision Transformer - Pytorch

link

Click the link

6.3 2. Install

— **These packages contain :**

— *Datasets and dataloaders*
— *Image processors for resizing and converting image into tensor*
— *Sample Visualisation*
— *Patch embedding*
— *Multihead Self-Attention Block*
— *Multi-Layer Perceptron Block*
— *Transformer Encoder Block*
— *ViT Assembled*
— *Utility modules (training, testing, calculating performance indicators, ploting . . .)*

Note : Please refer to the requirements.txt file for a list of necessary dependencies.

— **Installing packages**

```
!pip install git+https://github.com/MasrourTawfik/images_segmentation_prompt/tree/main/
↪ViT_Implementation.git
```

```
print(f'Using PyTorch version: {torch.__version__}')
device="cuda" if torch.cuda.is_available() else "cpu"
print(f'Using device: {device}')
```

6.4 3. Dataset preparation

6.4.1 Classification dataset

```
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)
!cp /content/gdrive/MyDrive/ViT_test/DogsCats.zip /content

os.makedirs('data', exist_ok=True)

!unzip -q DogsCats.zip -d /content/data

# Setup directory paths to train and test images
train_dir = '/content/data/DogsCats/train'
test_dir = '/content/data/DogsCats/test'
```

```
# Setup directory paths to train and test images
train_dir = r'folder_name\train'
test_dir = r'folder_name\test'
```

6.4.2 Datasets_DataLoaders

— **Use ImageFolder to create dataset(s)**

```
train_data = datasets.ImageFolder(train_dir, transform=transform)
test_data = datasets.ImageFolder(test_dir, transform=transform)
```

— **Get class names from folders**

```
class_names = train_data.classes
```

6.4.3 Resize_Tensorize

Note : Please refer to the class for more details

```
transformer=Resize_Tensorize()
transform=transformer.create_transforms()
```

6.4.4 Sample_Viz

```
sample_viz=Sample_Viz()
sample_viz.visualize_sample(train_dataloader, class_names)
```

6.5 4. ViT components

6.5.1 PatchEmbedding

*In this class we will turn image into patches and flattened the patches (from 2D to 1D vector)

— **Class initialization**

```
def __init__(self,
                in_channels:int=3, #RGB
                patch_size:int=16,
                embedding_dim:int=768):  # for each patch 16x16x3=768 every patch has 768
```

```
# Create a convolution 2D layer to turn an image into patches
self.patcher = nn.Conv2d(in_channels=in_channels,
                         out_channels=embedding_dim,
                         kernel_size=patch_size,
                         stride=patch_size,
                         padding=0)
```

```
# Create a layer to flatten the patch feature maps into a single dimension
self.flatten = nn.Flatten(start_dim=2, # only flatten the feature map dimensions into a␣
↪single vector
                          end_dim=3)
```

6.5.2 Positional Embedding

— **Class Embedding**

— **Position Embedding**

```
# Create learnable class embedding (needs to go at front of sequence of patch embeddings)
self.class_embedding = nn.Parameter(data=torch.randn(1, 1, embedding_dim),
                                    requires_grad=True)

# Create learnable position embedding
self.position_embedding = nn.Parameter(data=torch.randn(1, self.num_patches+1, embedding_
↪dim),
                                       requires_grad=True)
```

6.5.3 MultiheadSelfAttentionBlock

```
def __init__(self,
                embedding_dim:int=768, # Hidden size D from Table 1 for ViT-Base
                num_heads:int=12, # Heads from Table 1 for ViT-Base
                attn_dropout:float=0)
```

```
self.layer_norm = nn.LayerNorm(normalized_shape=embedding_dim)
```

```
self.multihead_attn = nn.MultiheadAttention(embed_dim=embedding_dim,
                                                    num_heads=num_heads,
                                                    dropout=attn_dropout,
                                                    batch_first=True)
```

6.5.4 MLPBlock

— **Initialization with Hyperparameters :**

```
def __init__(self,
             embedding_dim:int=768,
             mlp_size:int=3072,
             dropout:float=0.1)
```

— **Multilayer Perceptron (MLP) Layers :**

```
# 4. Create the Multilayer perceptron (MLP) layer(s)
        self.mlp = nn.Sequential(
            nn.Linear(in_features=embedding_dim,
                      out_features=mlp_size),
            nn.GELU(), # "The MLP contains two layers with a GELU non-linearity (section␣
↪3.1)."
            nn.Dropout(p=dropout),
            nn.Linear(in_features=mlp_size, # needs to take same in_features as out_
↪features of layer above
                      out_features=embedding_dim), # take back to embedding_dim
            nn.Dropout(p=dropout) # "Dropout, when used, is applied after every dense␣
```

(suite sur la page suivante)

(suite de la page précédente)

```
↪layer.."
        )
```

— **Forward Method :**

6.5.5 TransformerEncoderBlock

```
# 3. Create MSA block
        self.msa_block = MultiheadSelfAttentionBlock(embedding_dim=embedding_dim,
                                                     num_heads=num_heads,
                                                     attn_dropout=attn_dropout)

        # 4. Create MLP block
        self.mlp_block =  MLPBlock(embedding_dim=embedding_dim,
                                   mlp_size=mlp_size,
                                   dropout=mlp_dropout)
```

6.5.6 ViT

6.6 5. code source

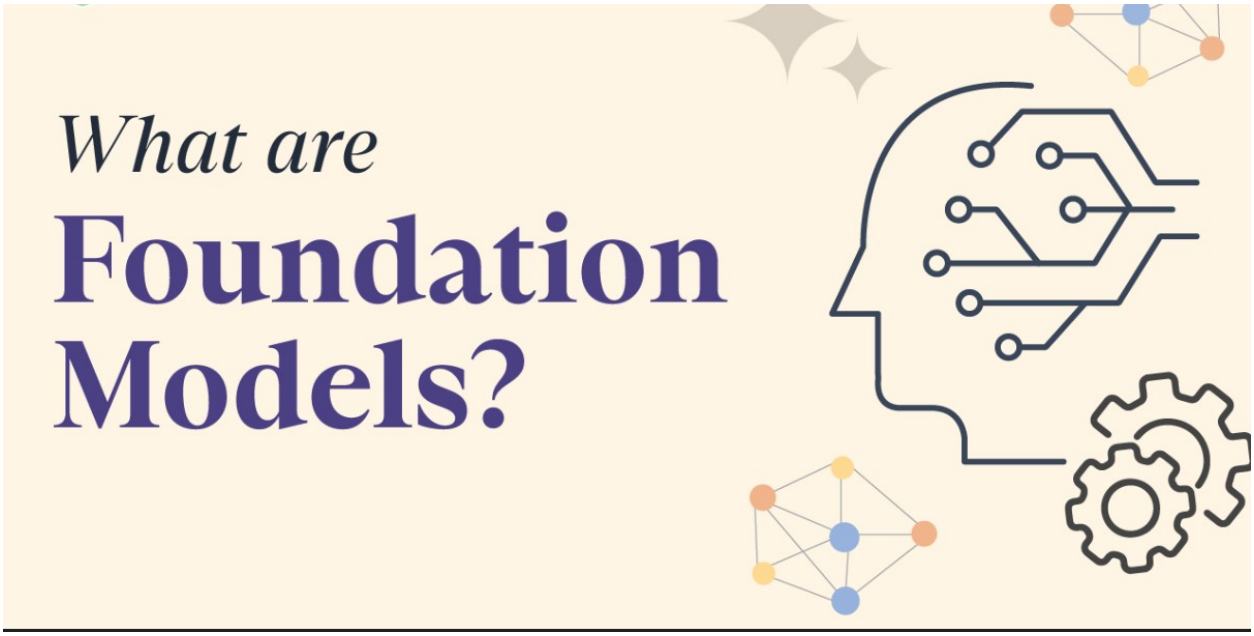
Link to github repository and colab applications

For more practice and to learn more, we can visit this tutorial.

— Find the link to github repository « project »
— Find the link to github repository « ViT_Implementation »
— Find the link to colab notebook

CHAPITRE 7

Definition



7.1 Introduction to Foundation models

7.2 What are Foundation Models ?

ChatGPT

Click here to know that how ChatGPT utilizes self-attention and encoding mechanisms to process user prompts and generate human-like responses.

7.3 History of Foundation Models

7.4 Types of Foundation Models

NLP

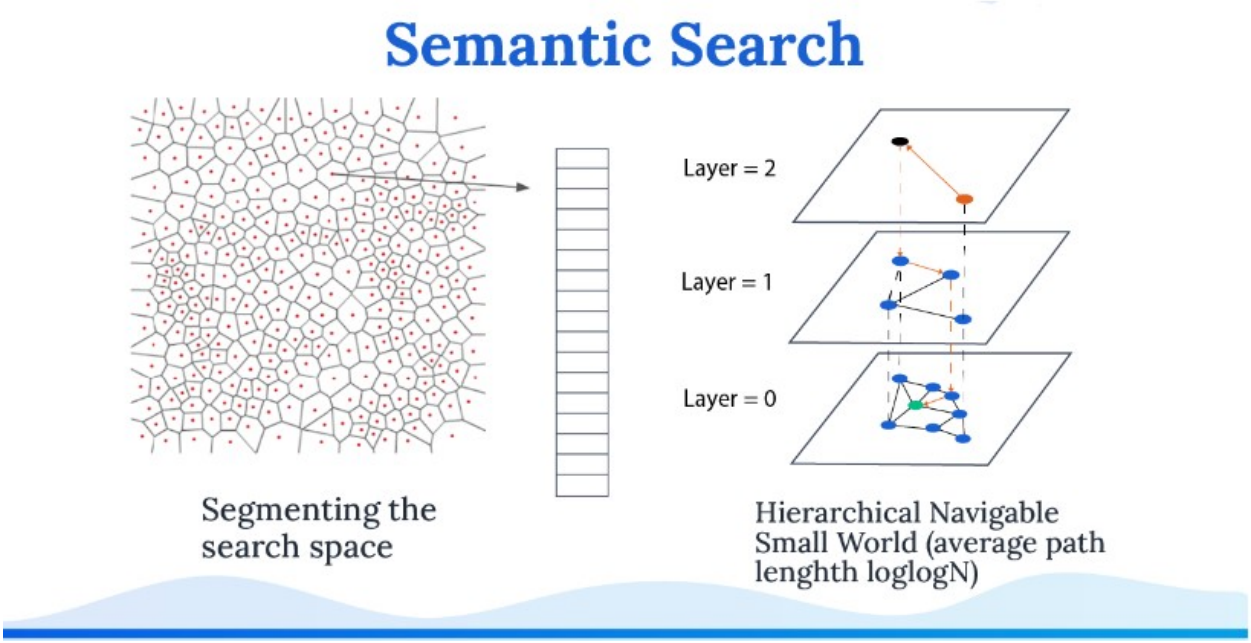
Natural language processing is a field of artificial intelligence that helps computers understand, interpret and manipulate human language.

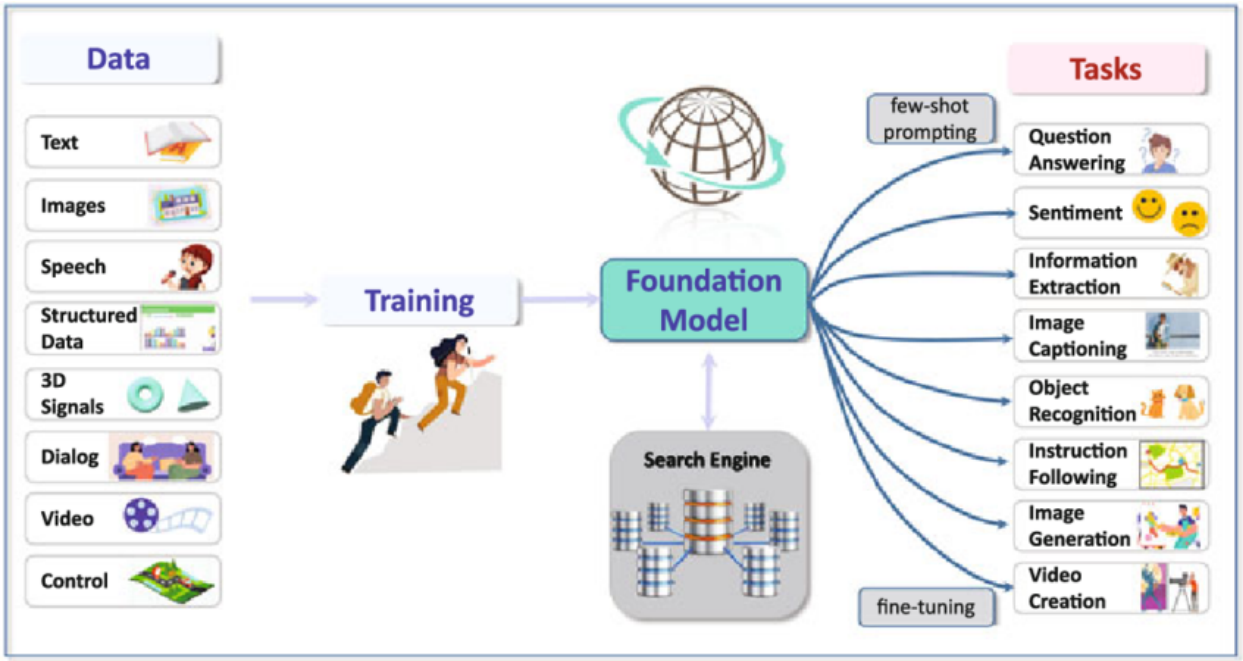
7.5 Applications of Foundation Models

— **Semantic Search**

7.6 Limitations of Foundation Models

7.7 Conclusion


Semantic Search
Layer = 2
Layer = 1
Layer = 0
Segmenting the search space
Hierarchical Navigable Small World (average path lenghth loglogN)


Data
Text
Images
Speech
Structured Data
3D Signals
Dialog
Video
Control
Training
Foundation Model
Search Engine
few-shot prompting
fine-tuning
Tasks
Question Answering
Sentiment
Information Extraction
Image Captioning
Object Recognition
Instruction Following
Image Generation
Video Creation

CHAPITRE 8

Famous Models

contents

8.1 Grounding DINO

8.1.1 1. Introduction

8.1.2 2. Grounding DINO Performance

GLIP T vs. Grounding DINO T speed and mAP comparison


Grounding DINO
State of the Art
Zero-Shot Object Detector


image1.jpg


piano guitar phone


image3.jpg


phone hat
hat 0.70
phone 0.40

8.1.3 3. Advantages of Grounding DINO

For more information

— Find the link to « Non-Maximum Suppression (NMS). »
— Find the link to « How to Code Non-Maximum Suppression (NMS) in Plain NumPy. »

8.1.4 4. Grounding DINO Architecture

Here is how Grounding DINO would work on this image :

8.1.5 5. Reference

source

— Find the link to « Grounded Language-Image Pre-training. »
— Find the link to « DINO: DETR with Improved DeNoising Anchor Boxes for End-to-End Object Detection »
— Find the link to « Non-Maximum Suppression (NMS). »
— Find the link to « How to Code Non-Maximum Suppression (NMS) in Plain NumPy. »

8.2 Segment Anyting Model

8.2.1 1. Introduction to SAM :

source

— Find the link to « SA-1B Dataset. »

8.2.2 2. What is the Segment Anything Model ?

source

— Find the link to « image segmentation »
— Find the link to « foundation models guide »

source

— Find the link to « segmentation masks »

8.2.3 3. SAM's network architecture

The Segment Anything (SA) project introduces a new task, model, and dataset for image segmentation

The architecture of the segment anything model (SAM). The SAM consists of the following components : An Image Encoder, a Decoder, and a Mask Decoder

source

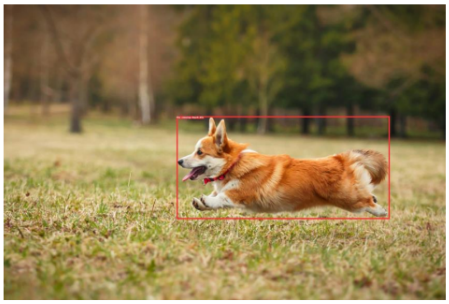
— Find the link to « NLP models »

source

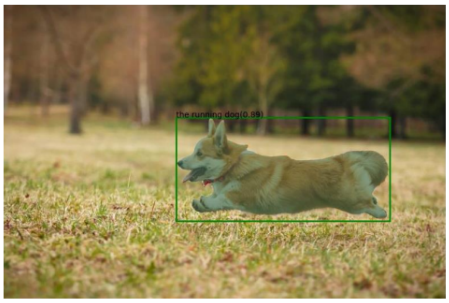
— Read more at « segment anything model sam explained »

Segment Anything Model

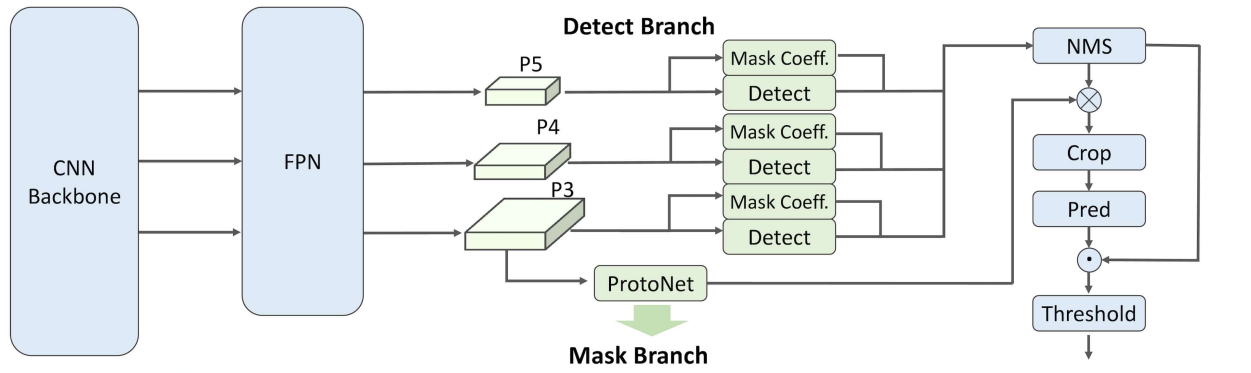
Text Prompt: The running dog

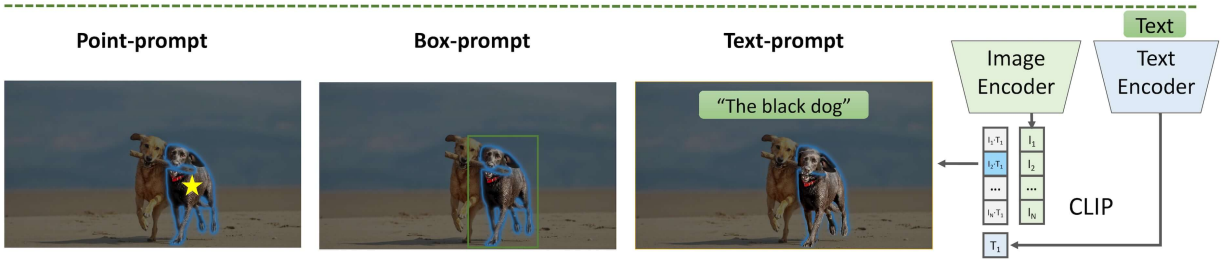


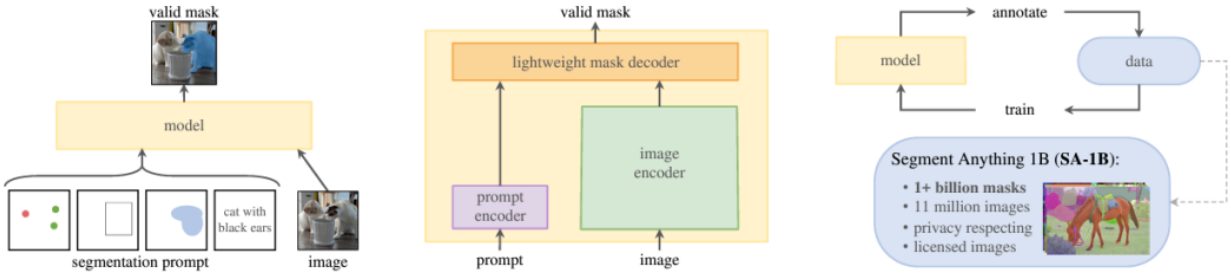
Grounding DINO: OpenSet Detection



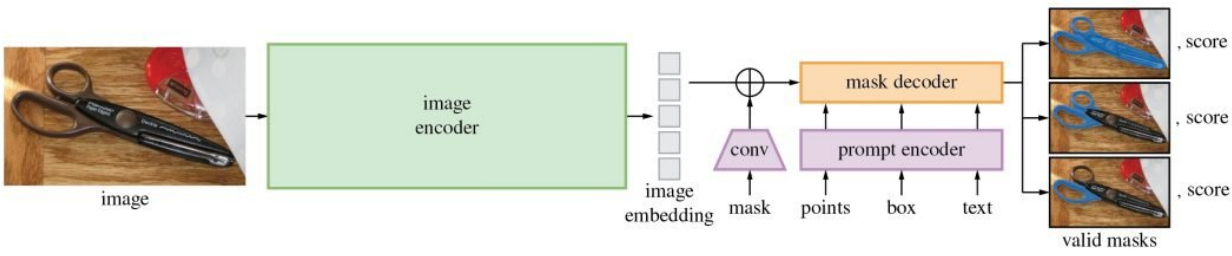
Grounded-SAM: OpenSet Detection and Segmentation







(a) **Task**: promptable segmentation (b) **Model**: Segment Anything Model (**SAM**) (c) **Data**: data engine (top) & dataset (bottom)



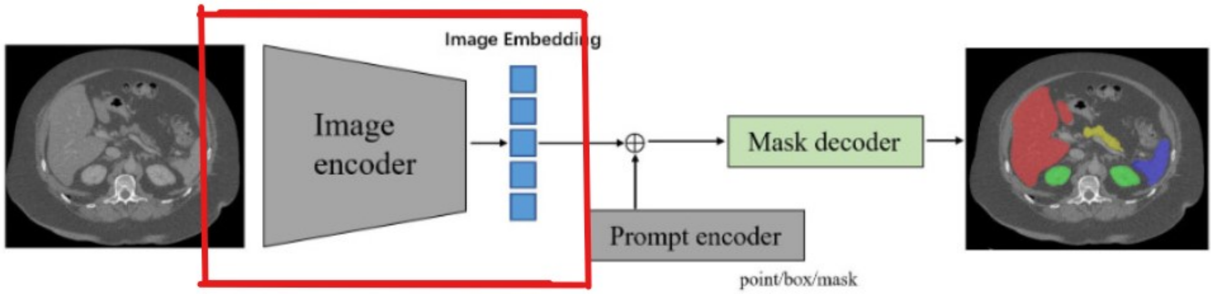


Figure 2: Overview of the architecture of Segment Anything Model (SAM).

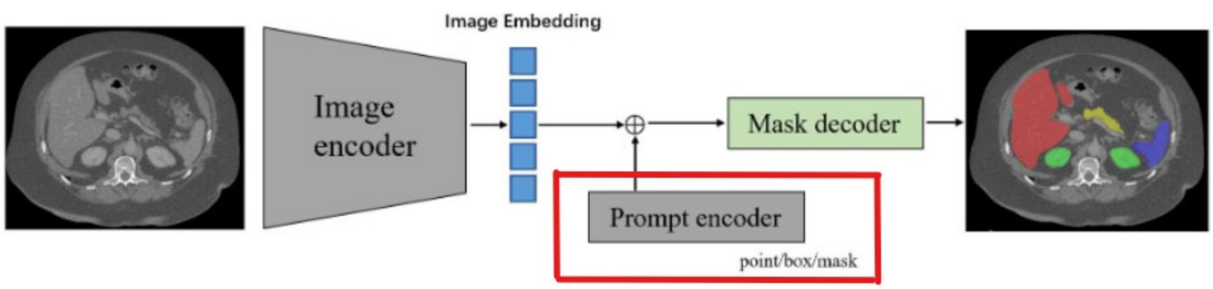


Figure 2: Overview of the architecture of Segment Anything Model (SAM).

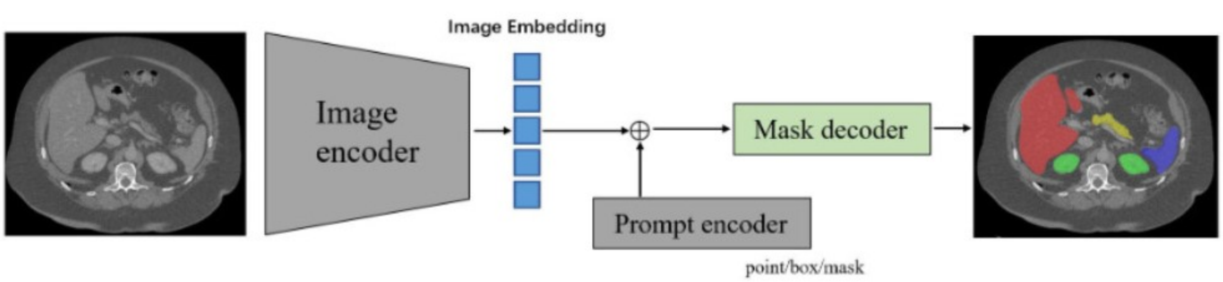
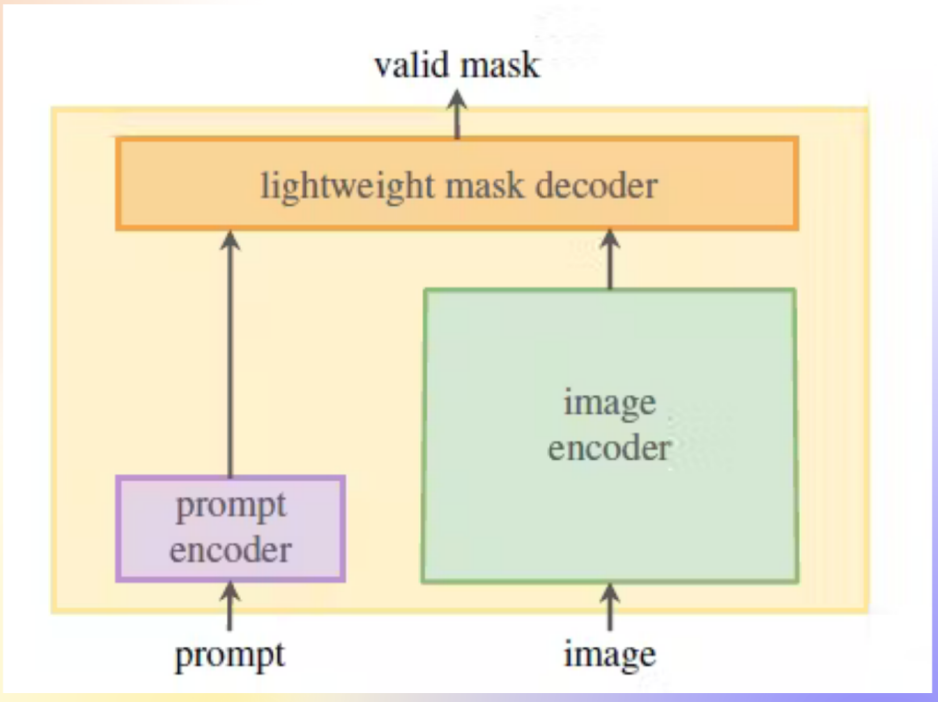


Figure 2: Overview of the architecture of Segment Anything Model (SAM).



screenshot from SAM paper

1001epochs.co.uk

8.2.4 4. How does SAM support real-life cases ?

— **Versatile segmentation :**

— **Zero-Shot Transfer :**

Real-Time Interaction :

Multimodal Understanding :

Efficient Data Annotation :

Equitable Data Collection :

Content Creation and AR/VR :

Scientific Research :

Overall

— *SAM's versatility, adaptability, and real-time capabilities make it a valuable tool for addressing real-life image segmentation challenges across diverse industries and applications.*

8.2.5 5. Reference

---

source

— Find the link to « segment anything model sam explained »
— Find the link to « segment anything model sam paper »
— Find the link to « SA-1B Dataset. »
— Find the link to « image segmentation »
— Find the link to « foundation models guide »
— Find the link to « segmentation masks »
— Find the link to « NLP models »
— Find the link to « segment anything model »

CHAPITRE 9

Groundingsam project implemention

9.1 Cloning the Repository

```
!git clone https://github.com/SAAD1190/GroundingSam.git
```

9.2 Setting the Working Directory

```
HOME = "/content/GroundingSam"
```

9.3 Navigating to the Directory

```
%cd {HOME}
```

9.4 Installing Dependencies

```
!bash dependencies.sh # Install the necessary dependencies (Ignore the pip dependency)
```

9.5 Creating Directories for Models and Annotations

```
!mkdir {HOME}/weights
!mkdir {HOME}/annotations
```

9.6 Downloading GroundingDINO Model Weights

```
%cd ./weights
!wget -q https://github.com/IDEA-Research/GroundingDINO/releases/download/v0.1.0-alpha/
↪groundingdino_swint_ogc.pth
```

9.7 Returning to the Main Directory

```
%cd {HOME}
```

9.8 Installing Segment Anything via pip

```
!pip install 'git+https://github.com/facebookresearch/segment-anything.git'
```

9.9 Downloading SAM Model Weights

```
%cd ./weights
!wget -q https://dl.fbaipublicfiles.com/segment_anything/sam_vit_h_4b8939.pth
```

9.10 Returning to the Main Directory (Again)

```
%cd {HOME}
```

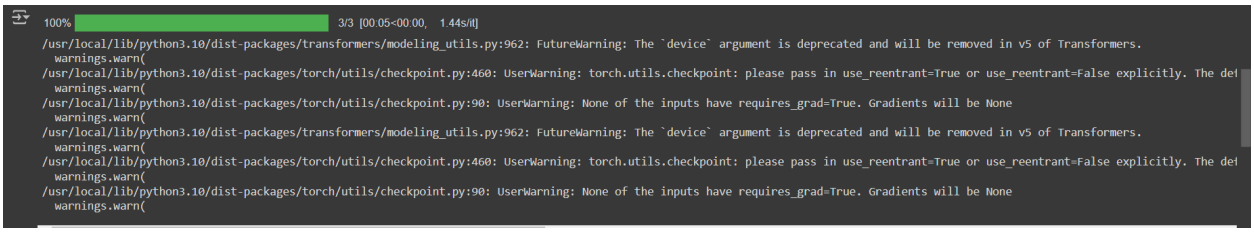
9.11 Importing and Initializing

```
from GroundingSam import *
classes = ['piano', 'guitar', 'phone', 'hat']
groundingsam = GroundingSam(classes=classes)
```

9.12 Detection, Annotation, and Segmentation

```
from GroundingSam import *
detections = groundingsam.get_detections()
groundingsam.annotate_images()
groundingsam.get_masks()
```

```
from GroundingSam import *
detections = groundingsam.get_detections()
```




```
groundingsam.annotate_images()
```

```
groundingsam.get_masks()
```

9.13 Summary

image1.jpg

piano guitar phone

image3.jpg

phone hat

image2.jpg

piano guitar

image1.jpg

piano guitar phone

image3.jpg

phone hat

image2.jpg

piano guitar

CHAPITRE 10

State Of The Art

CHAPITRE 11

Prompt Generator / Analyzer

contents

11.1 Overview

11.2 Prompt Gemini Generator

Parameters :

Operations :

Prompt Generation (generate_prompts) :

11.3 Features

11.3.1 Prompt pre-processing

```
def prompt_processing(self)
```

— **Core Functionality :**

— **Output**

11.3.2 Similarity Reduction

```
def prompts_similarity(self, remove_similar=False, threshold=0.7):
```

— **Functionality :**

— **Parameters :**

— **Output**

— **Use Case :**

11.3.3 Complexity Analysis

```
def prompt_complexity(self):
```

— **Functionality :**

— **Output**

— **Use Case :**

11.3.4 Readability Analysis

```
def readability(self):
```

— **Functionality**

— **Output**

— **Use Case**

11.4 Prompt processing

— **Functionality**

— **Parameter :**

— **Output :**

— **Use Case**

11.5 Prompt Generator Example

```
API_Key=input("Enter your API Key")
prompts=prompt_generator('gemini-pro-vision',API_Key)
prompts_dict=prompts.generate_prompts(number_of_prompts=10)
```

11.6 Prompt Analyzer Exemple

```
prompts_dict = {
    'image1.jpg': ['An early morning', 'Sunrise at the beach', 'Dawn breaks over the␣
↪ocean']
}
analyzer = prompt_analyzer(prompts_dict)
analyzer.process_prompts(complexity=True)
```

CHAPITRE 12

Transformer : Attention Is All You Need

12.1 1. Introduction

12.2 2. Background

12.3 3. Self-Attention Mechanism

12.4 4. Multi-Head Self-Attention

12.5 5. Position-Wise Feed-Forward Networks

12.6 6. Transformer Model

12.7 7. Attention Visualization

12.8 8. Experimental Results

12.9 9. Conclusion

12.10 Summary

For more information

— « self-attention-from-scratch »
— You can view more by clicking the link to the paper « Attention is all you need »
— or simply clicking the picture

CHAPITRE 13

AN IMAGE IS WORTH 16X16 WORDS

13.1 1. Objectives of the Paper

13.2 2. Paper Contributions

13.3 3. Paper Limitations, Further Research

13.4 Summary

For more information

— You can view more by clicking the link to the paper « An Image is Worth 16x16 Words: »
— or simply clicking the picture

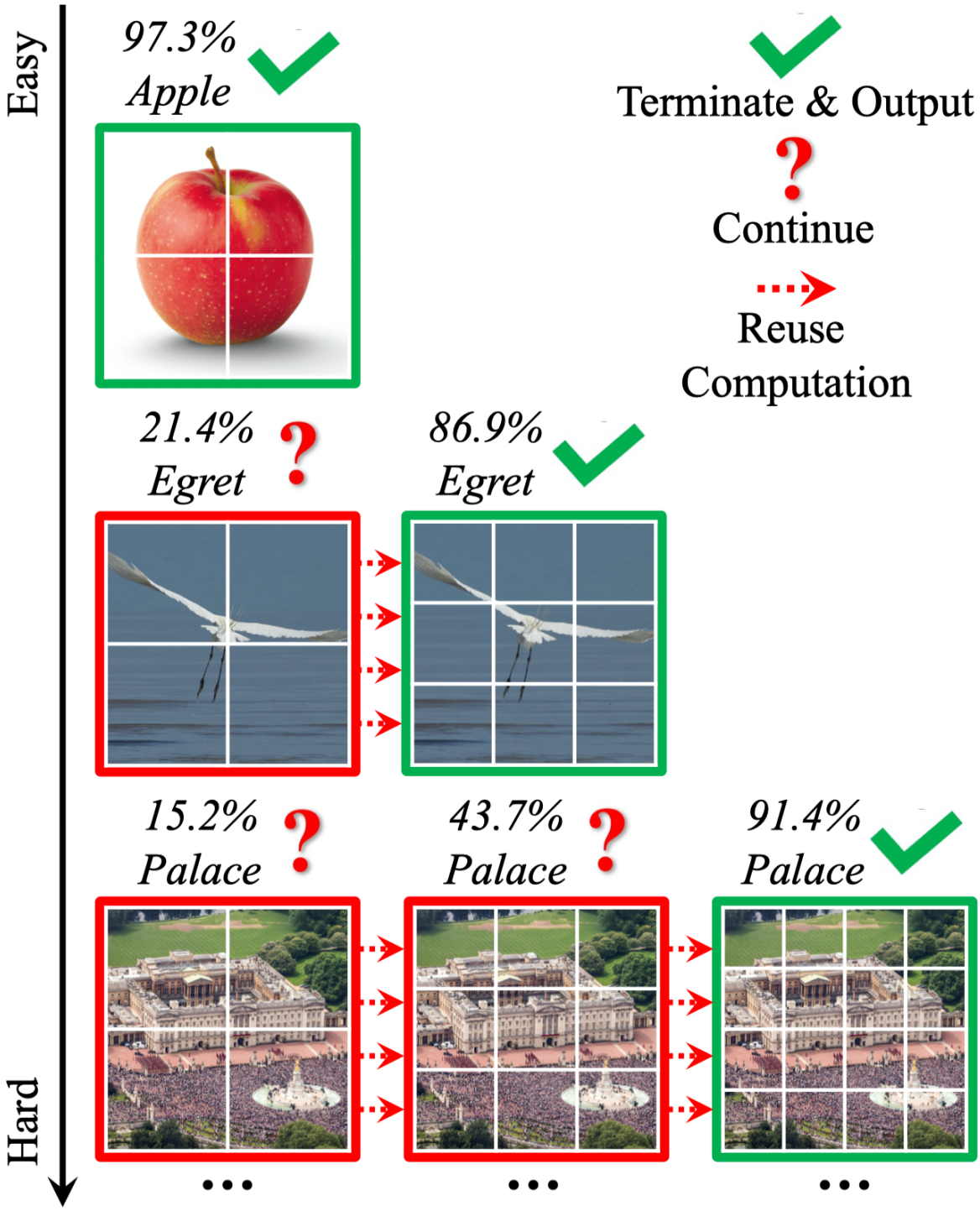


Figure 1: Examples for DVT.

CHAPITRE 14

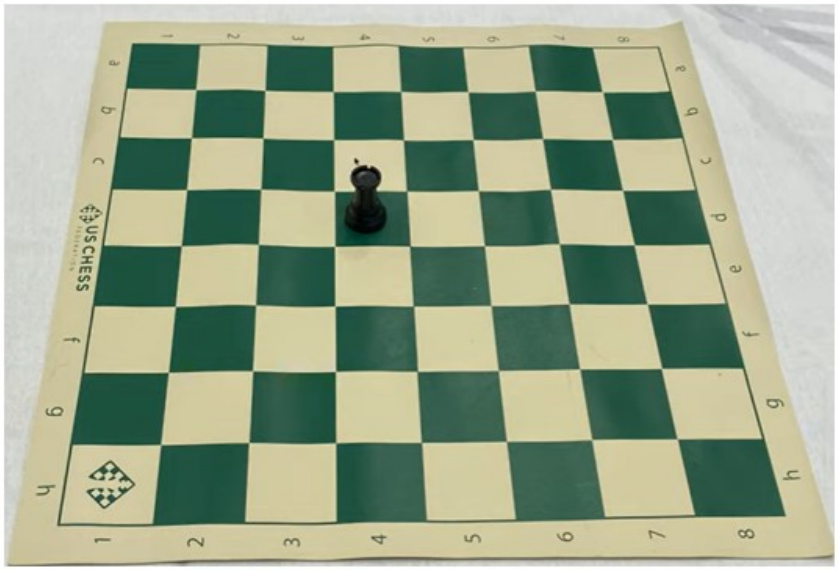
PASCAL VOC

<span style= »color :white ; »>”

14.1 1. Introduction

14.2 2. One annotation :

[image_0320.jpg]



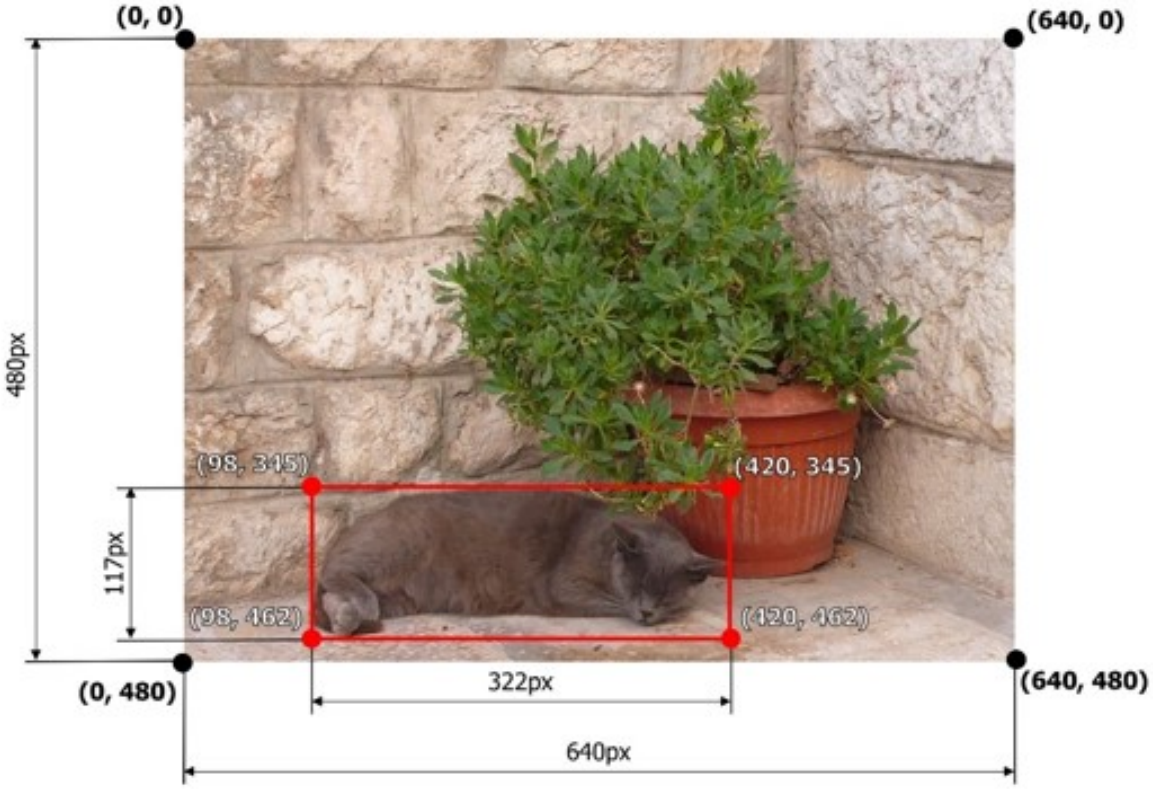
Annotation pascal voc format xml file

[Image_0320.xml]

IMG_0320.xml

```
<annotation>
    <folder>Disrupt 1</folder>
    <filename>IMG_0320.JPG</filename>
    <size>
        <width>2284</width>
        <height>1529</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>black-rook</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <occluded>0</occluded>
        <difficult>0</difficult>
        <bndbox>
            <xmin>957</xmin>
            <ymin>452</ymin>
            <xmax>1071</xmax>
            <ymax>635</ymax>
        </bndbox>
    </object>
</annotation>
```

14.3 3. Multiple annotations :



IMG_0159.xml

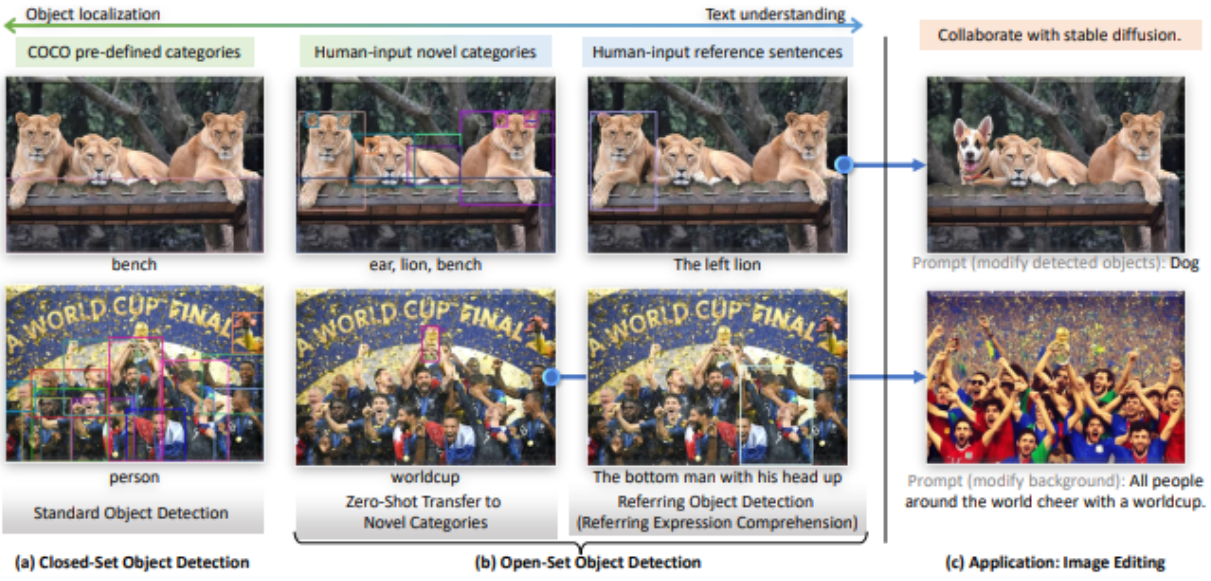
```
<annotation>
    <folder>Disrupt 1</folder>
    <filename>IMG_0159.JPG</filename>
    <size>
        <width>2284</width>
        <height>1529</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>black-queen</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <occluded>0</occluded>
        <difficult>0</difficult>
        <bndbox>
            <xmin>1708</xmin>
            <ymin>332</ymin>
            <xmax>1878</xmax>
```

14.4 4. Additional fields:

14.5 5. Conclusion

CHAPITRE 15

Grounding DINO



15.1 Introduction :

15.2 Main Concept :

15.3 Advantages :

15.4 Existing Approaches :

15.5 Performance :

15.6 Contributions :

15.7 General Conclusion of the Paper

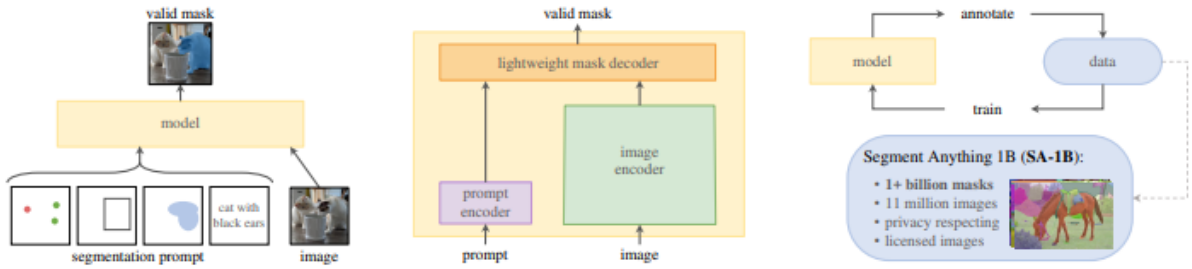
— **« Grounding DINO : Marrying DINO with Grounded Pre-Training for Open-Set Object Detection » :**

For more information

— You can view more by clicking the link to the paper « Grounding DINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection »

CHAPITRE 16

Segment Anything



(a) **Task**: promptable segmentation (b) **Model**: Segment Anything Model (**SAM**) (c) **Data**: data engine (top) & dataset (bottom)

16.1 Introduction

16.2 Key Components

16.2.1 1. Task : Promptable Segmentation

16.2.2 2. Model : Segment Anything Model (SAM)

16.2.3 3. Data : SA-1B Dataset

16.3 Methodology

16.4 Experiments and Results

16.5 Conclusion

For more information

— You can view more by clicking the link to the paper « Segment Anything »