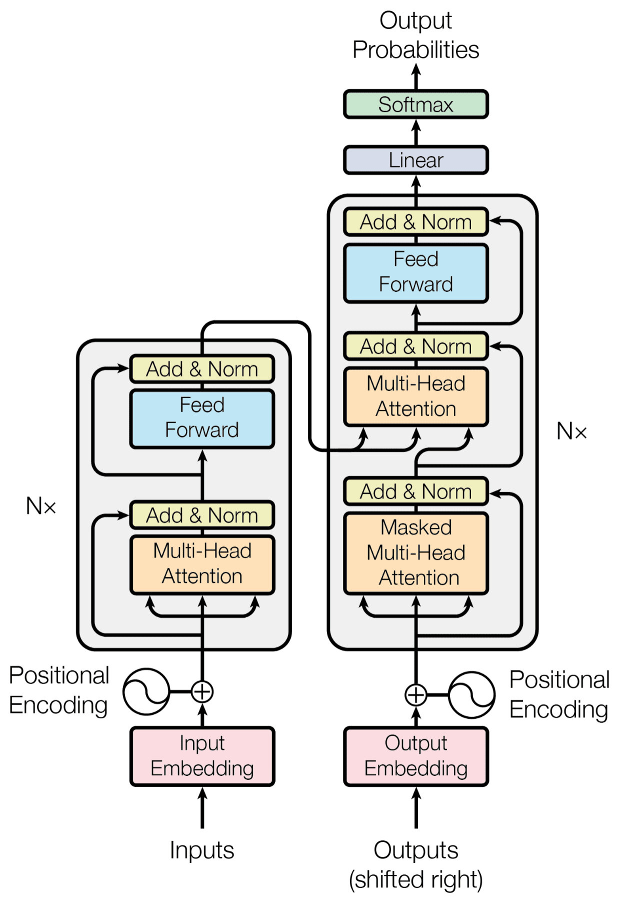
Research and analyze Transformer:

1) Learn Transformer based Encoder models and GPT models .

521K0126_521K0133_DEEP_LEARNING

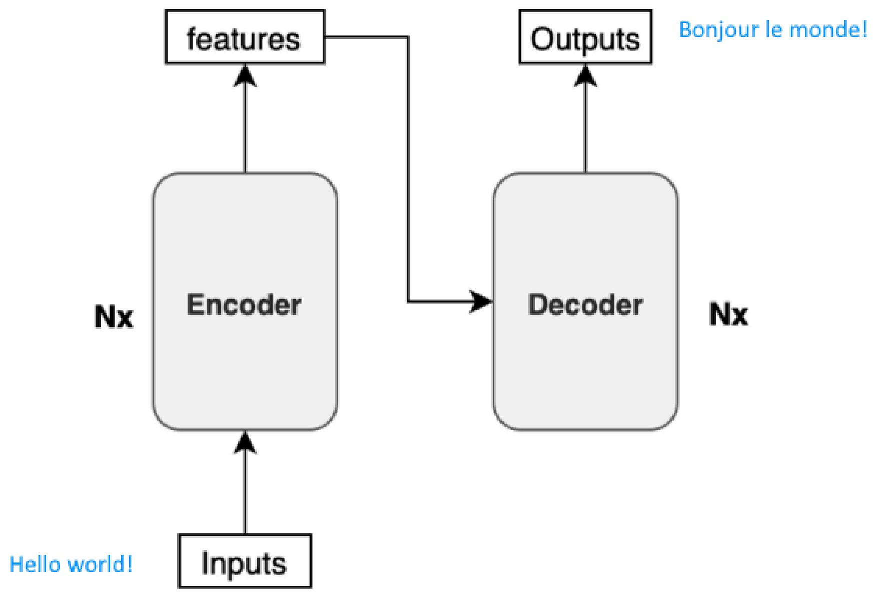
Transformer Architecture

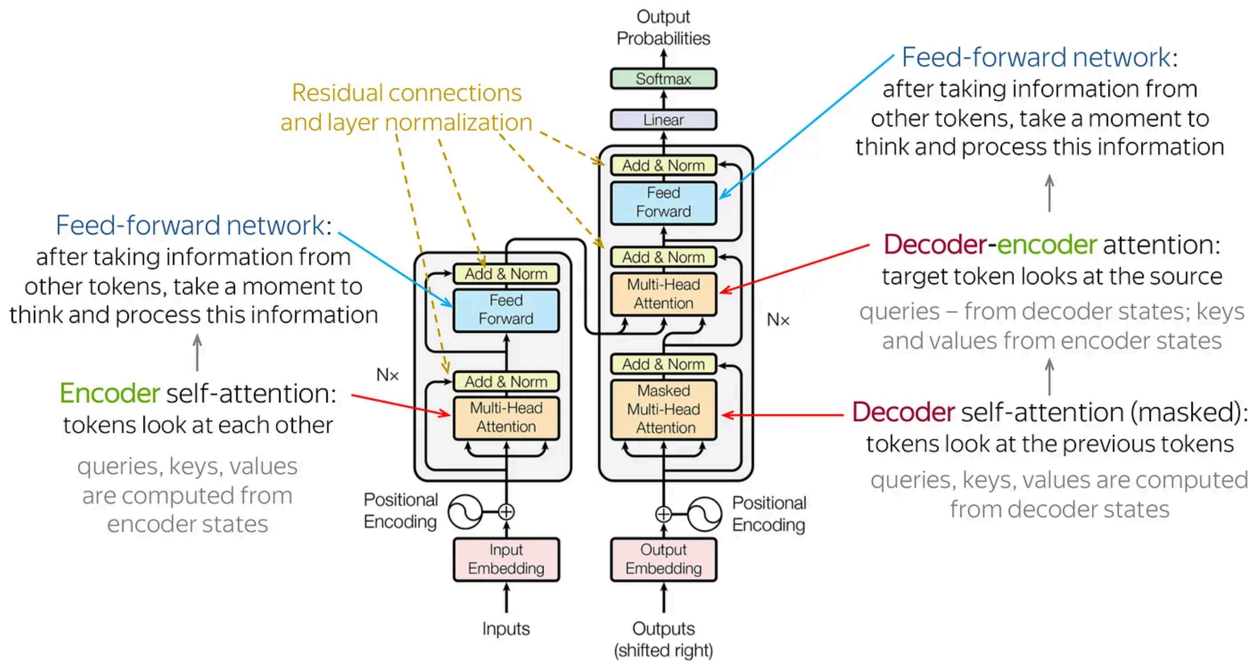


An Encoder-Decoder Architecture

In short, the architecture is simple because it has no recurrence and convolutions. It uses other common concepts like an encoder-decoder architecture, word embeddings, attention mechanism, softmax, and so on.

The transformer itself is an encoder-decoder network at a high level. In fact, the original transformer published in the “Attention is all you need” paper is a neural machine translation model. For example, we can train it to translate English into French sentences




Output Probabilities
Softmax
Linear
Residual connections and layer normalization
Feed-forward network: after taking information from other tokens, take a moment to think and process this information
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
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
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
Nx
Positional Encoding
Input Embedding
Inputs
Positional Encoding
Output Embedding
Outputs (shifted right)

The task of the encoder, on the left half of the Transformer architecture, is to map an input sequence to a sequence of continuous representations, which is then fed into a decoder. The decoder, on the right half of the architecture, receives the output of the encoder together with the decoder output at the previous time step to generate an output sequence. The paper uses "Nx" to indicate that there are multiple blocks in the encoder and decoder architecture. The N represents the number of identical blocks stacked together in either the encoder and decoder. For instance, an encoder-decoder architecture with N=2 would have 2 encoder blocks followed by two decoder blocks.

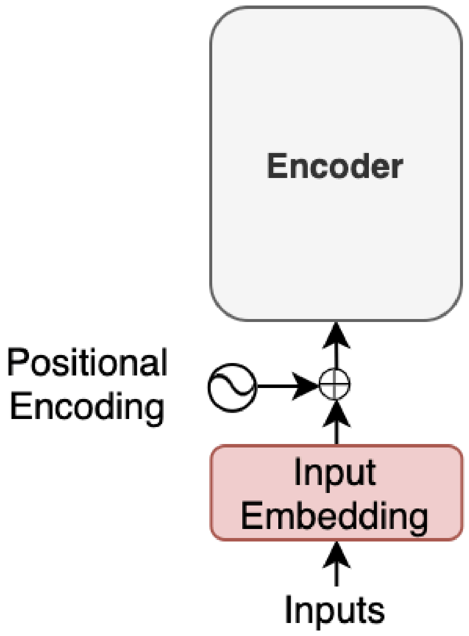
This allows the model to learn more complex relationships within the sequences by processing the information through multiple layers.

Input Embedding and Positional Encoding

Like all neural translation models, we typically tokenize an input sentence into separate tokens. This tokenized sentence forms a fixed-length sequence. For example, if the maximum length is 200, every sentence will consist of 200 tokens with trailing paddings. This intuitive representation would appear as follows:
('Hello', 'world', '!', <pad>, <pad>, …, <pad>)

These tokens are usually integer indices within a vocabulary dataset. Therefore, it could be a sequence of numbers, such as the following:
(8667, 1362, 106, 0, 0, …, 0)
The number 8667 corresponds to the token "Hello" in this example. It could also include special characters like <EOS> (end-of-sentence marker), depending on your specific tokenizer and vocabulary dataset. If you're using a model from Hugging Face, there's a dedicated tokenizer that manages these details. However, if you're building a model from scratch, you'll need to decide how to tokenize a sentence, establish a vocabulary dataset, and assign an index to each token.

To input these tokens into the neural network, each token is converted into an embedding vector, a common practice in neural machine translation and other natural language models. In the paper, they utilize a 512-dimensional vector for such embeddings. Therefore, if the maximum sentence length is 200, the shape of each sentence will be (200, 512). The transformer learns these embeddings from scratch during training. If you're unfamiliar with word embeddings, they essentially represent words or tokens as dense vectors in a continuous vector space.

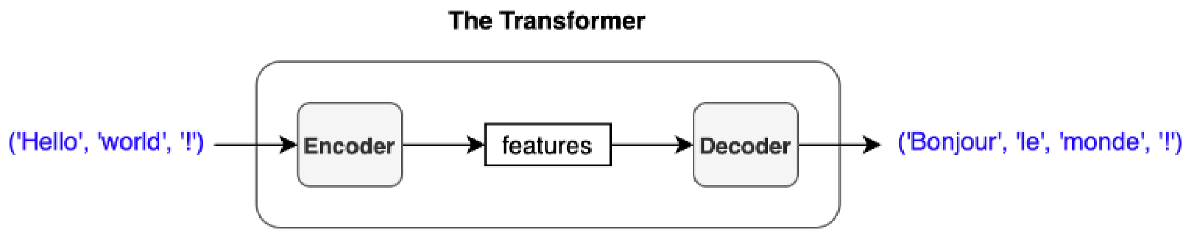


Furthermore, in the transformer architecture, positional encoding is added to each embedding to enable the model to understand word positions without using recurrence. For more information on positional encoding, please refer to this article.

The key point here is that the input to the transformer isn't the individual characters of the input text, but rather a sequence of embedding vectors. Each vector encapsulates the semantics and position of a token. Through multiple encoder blocks, the encoder conducts linear algebra operations on these vectors to extract contextual information for each token from the entire sentence and enhance the embedding vectors with pertinent information for the target task.

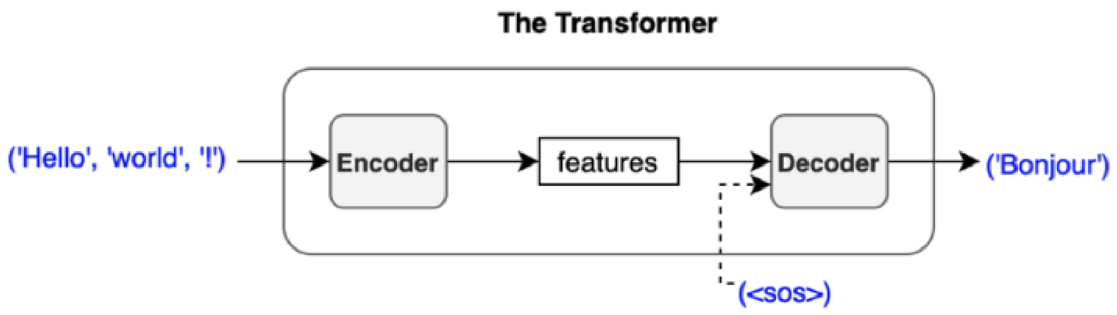
Softmax and Output Probabilities

The decoder utilizes input features from the encoder to generate an output sentence. These input features are essentially the enriched embedding vectors obtained from the encoder.

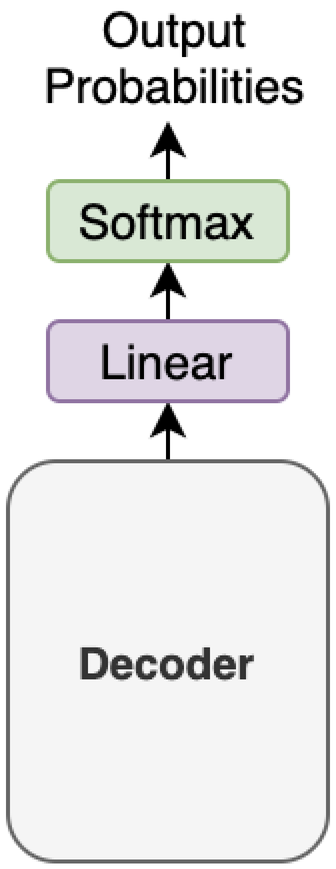


For simplicity, I represent a sentence as ('Hello', 'world', '!'), but in reality, the inputs to the encoder are input embeddings with positional encodings.
The decoder generates one token at a time, where each output token becomes the subsequent input to the decoder. This process, known as "auto-regressive," is a typical pattern for generating sequential outputs and is not unique to the transformer architecture. It enables a model to produce an output sentence of varying lengths compared to the input.

At the beginning of a translation, there is no previous output available. Therefore, we pass the start-of-sentence marker <SOS> (also known as the beginning-of-sentence marker <BOS>) to the decoder to initialize the translation process.



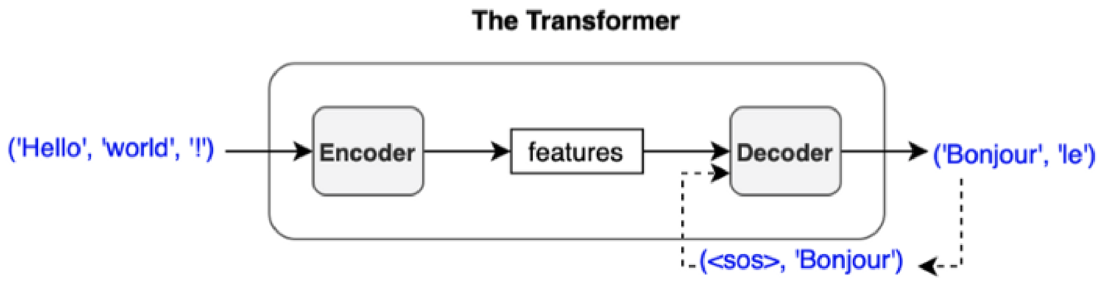
The decoder employs multiple decoder blocks to enhance the <SOS> token with contextual information derived from the input features. In essence, the decoder transforms the embedding vector <SOS> into a vector containing pertinent information for generating the first translated word (token).
Subsequently, the output vector from the decoder undergoes a linear transformation, adjusting the dimensionality of the vector from the embedding vector size (512) to the size of the vocabulary (for example, 10,000). The softmax layer then further converts the vector into probabilities across the 10,000 vocabulary items.



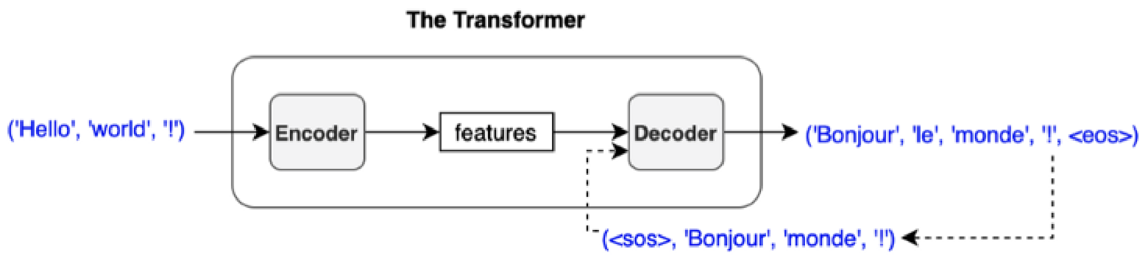
In this example, with a small vocabulary dataset containing only three words: 'like', 'cat', and 'I', suppose the model predicts the probabilities for the first output token as [0.01, 0.01, 0.98]. In this scenario, the word 'I' has the highest probability (0.98). So, according to the greedy method, we would choose the word 'I'.
However, it's essential to note that this choice depends on the method employed. This article assumes the greedy method, which selects the most probable word (i.e., the word with the highest probability) at each step.

Another approach, known as beam search, may offer improved performance in terms of the BLEU score. Unlike the greedy method, beam search considers multiple potential token sequences and selects the combination with the highest overall probability. In the paper, a beam search with a beam size of 4 is utilized.

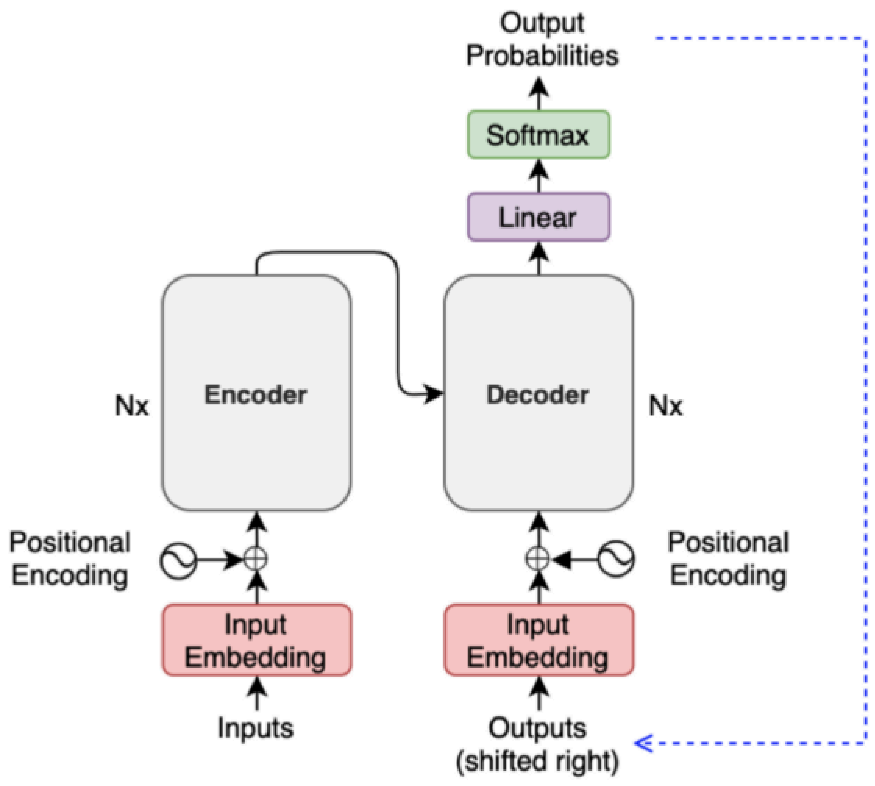
Regardless of the method used, the predicted word is fed back to the decoder to generate the next token. As illustrated below, we supply the chosen token to the decoder as the final part of the subsequent decoder input.



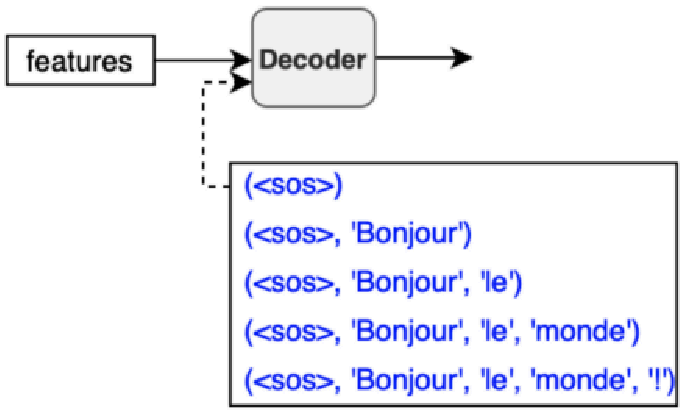
If 'Bonjour' is chosen as the first output token, the next input to the decoder will be (<SOS>, 'Bonjour'). Subsequently, we may receive ('Bonjour', 'le') as the output. This process continues iteratively until the model predicts the end-of-sentence marker <EOS> as the most probable output.


The Transformer
('Hello', 'world', '!')
Encoder
features
Decoder
('Bonjour', 'le', 'monde', '!', <eos>)
(<sos>, 'Bonjour', 'monde', '!')

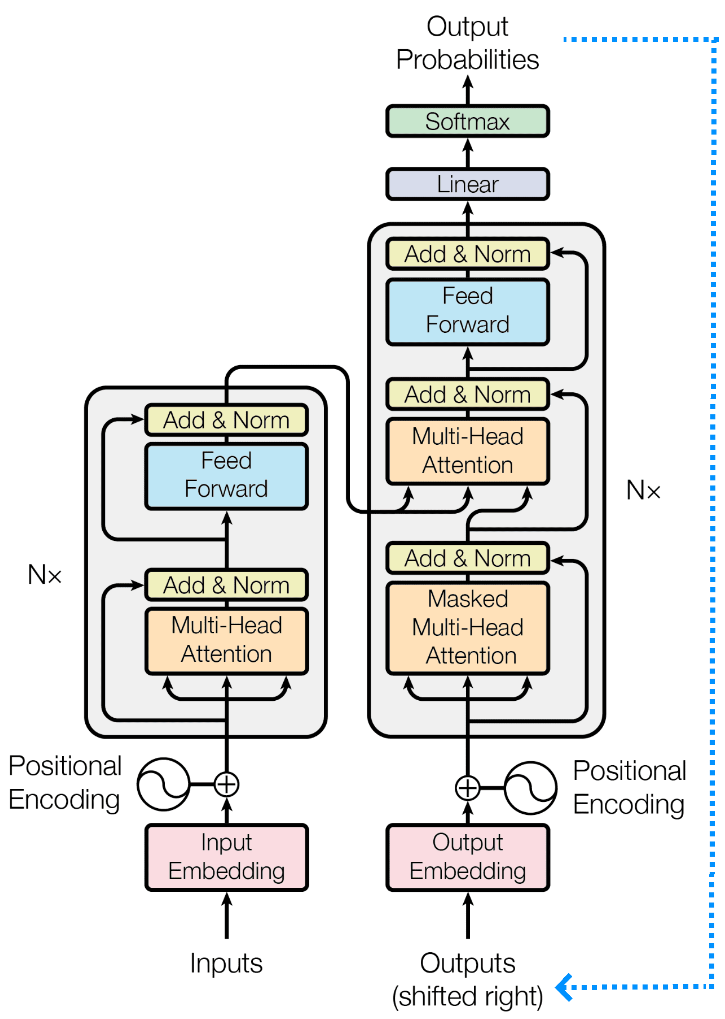
When we pass sequences like (<SOS>, 'Bonjour'), (<SOS>, 'Bonjour', 'le'), and so on, they are not of variable length. Similar to the encoder input, an input to the decoder consists of a sequence of tokens of fixed length. Additionally, each token is converted to an embedding vector through the embedding layer. Therefore, we're not passing characters or integer indices to the decoder, which mirrors the process in the encoder. We append an output token to the next input to the decoder. Consequently, the first output token becomes the second input token because the first input is <SOS>. This process is illustrated in the diagram as **'Outputs (shifted right)'.**



While generating one output at a time may seem slow, particularly during training, we typically employ the teacher-forcing method to enhance stability. With teacher-forcing, we feed label tokens (instead of predicted ones) to the decoder, facilitating more stable learning. This approach also accelerates training by enabling the preparation of an attention mask, which permits parallel processing.


features
Decoder
(<sos>)
(<sos>, 'Bonjour')
(<sos>, 'Bonjour', 'le')
(<sos>, 'Bonjour', 'le', 'monde')
(<sos>, 'Bonjour', 'le', 'monde', '!')

The diagram below illustrates the topics we've covered thus far:



To finally grasp the complete architecture of this transformer, we only need to delve into the internal workings of both the encoder and decoder blocks.

As we've known, this Transformer model (both the Encoder and Decoder blocks) relies solely on the use of self-attention, where the representation of a sequence (or sentence) is computed by relating different words in the same sequence, so now we will do a quick recap.

The Transformer Attention

Illustration

Essentially, the attention function can be seen as a mapping from a query and a set of key-value pairs to an output. The output is calculated as a weighted sum of the values, with each value's weight determined by a compatibility function of the query with the corresponding key.
The main components used by the Transformer attention are the following:

- q and k denoting vectors of dimension, d_k, containing the queries and keys, respectively
- v denoting a vector of dimension, d_v, containing the values
- Q, K, and V denoting matrices packing together sets of queries, keys, and values, respectively.
- W^Q, W^K and W^V denoting projection matrices that are used in generating different subspace representations of the query, key, and value matrices
- W^O denoting a projection matrix for the multi-head output

A quick note of what is Q,K and V
Query (Q): This vector represents the "question" being asked about a specific element in the sequence. It's like focusing on a particular word and trying to understand its meaning in relation to others.

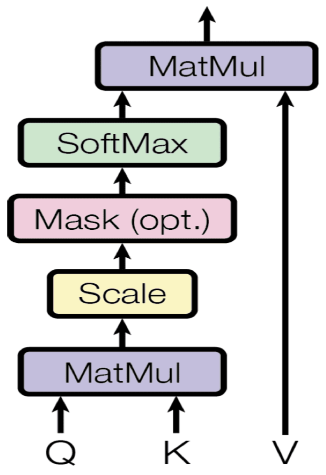
Key (K): The Key vector acts as a reference point for the query. It captures the essential information of each element in the sequence. Imagine it as a label or identifier that helps determine how relevant each element is to the query.

Value (V): This vector holds the actual content or information associated with each element. It's like the detailed description you get once you identify the relevant element based on the query and key.

https://medium.com/@zaiinn440/attention-is-all-you-need-the-core-idea-of-the-transformer-bbfa9a749937

Scaled Dot-Product Attention

The Transformer employs a scaled dot-product attention mechanism, which follows the general procedure of the attention mechanism described earlier.
As the name suggests, scaled dot-product attention first calculates a dot product for each query, q, with all of the keys, k. It then divides each result by √dk and applies a softmax function. This process yields the weights used to scale the values, v.



In practice, the computations performed by the scaled dot-product attention can efficiently handle the entire set of queries simultaneously. To achieve this, the matrices Q, K, and V are provided as inputs to the attention function.

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Here is step-by-step procedure for computing the scaled-dot product attention in theory:

Linear Transformations: We have three sets of vectors for each element in the sequence: Query (Q), Key (K), and Value (V). These are obtained by applying linear transformations to the original input sequence.

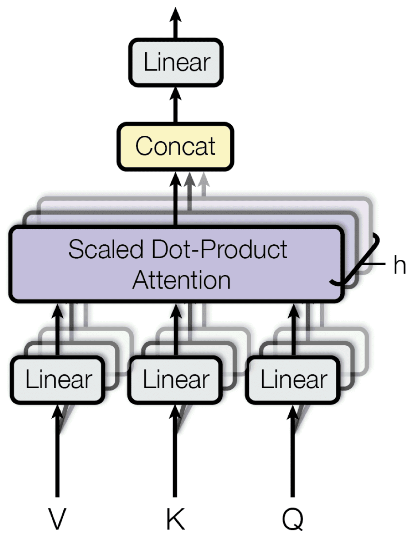
Dot Product: The model calculates the dot product between the query vector and each key vector in the sequence. This measures the raw similarity between the query and each element.

Scaling: To prevent exploding gradients during training, the dot products are divided by the square root of the dimension of the key vectors. This scaling helps stabilize the learning process.

Softmax: A softmax function is applied to the scaled dot products. This converts them into weights between 0 and 1, where a higher weight signifies greater relevance to the query.

Weighted Values: The attention weights are then multiplied by the corresponding value vectors for each element. This emphasizes the information from relevant elements based on the query.

Multi-Head Attention



Multi-head attention builds upon the concept of scaled dot-product attention. Here's the key difference:

- Multiple Heads: Instead of performing a single scaled dot-product attention, multi-head attention uses h independent sets of learnable Q, K, and V projections. Each head essentially learns to attend to different aspects of the relationship between elements.

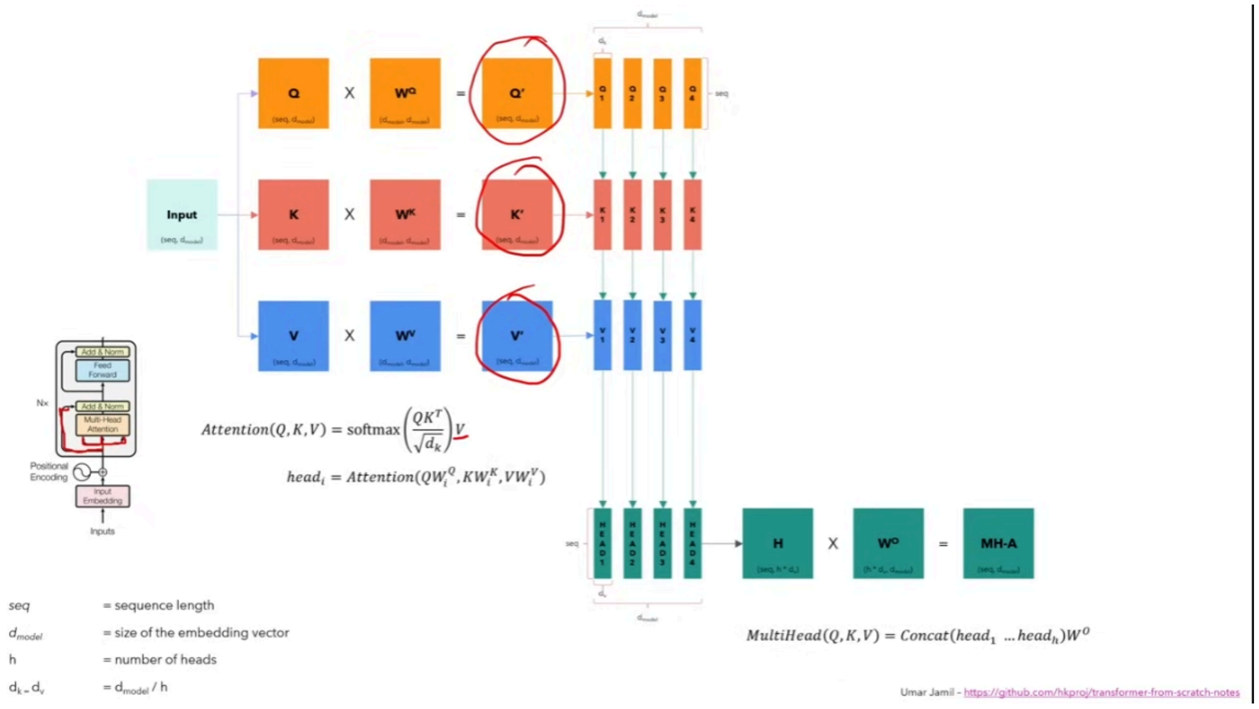
- Parallel Attention: Each set of Q, K, V vectors goes through its own scaled dot-product attention mechanism, resulting in h separate attention outputs.

- Concatenation: These h outputs are then concatenated to form a single vector.

- Linear Projection: Finally, a linear transformation is applied to this concatenated vector to produce the final output.

This multi-head approach allows the model to learn diverse attention patterns simultaneously. It can focus on some elements for their grammatical role, while others for their semantic meaning, leading to a richer understanding of the sequence.

For example, you're analyzing a sentence with multiple heads. One head might focus on verb conjugation, another on noun phrases, and another on sentiment. By combining these independent analyses, you gain a more comprehensive understanding of the sentence's structure and meaning.
In essence, scaled dot-product attention provides the foundation for calculating attention weights, while multi-head attention leverages this concept with multiple "heads" to learn richer and more diverse relationships within a sequence.
Note:



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

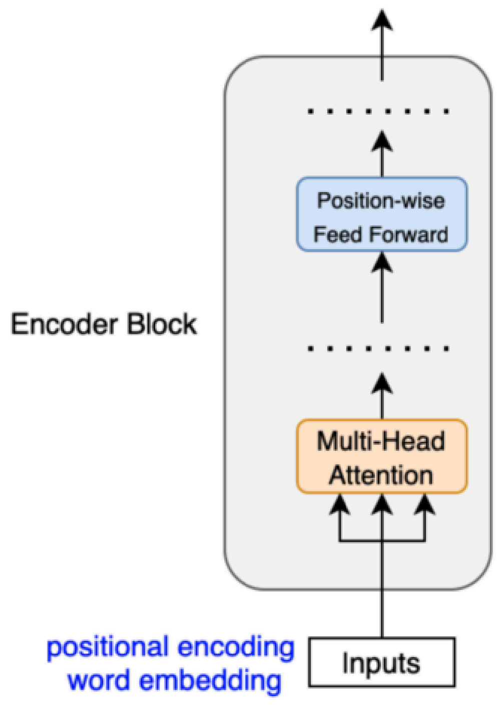
$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

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

seq	= sequence length
d_{model}	= size of the embedding vector
h	= number of heads
$d_k = d_v$	= d_{model} / h

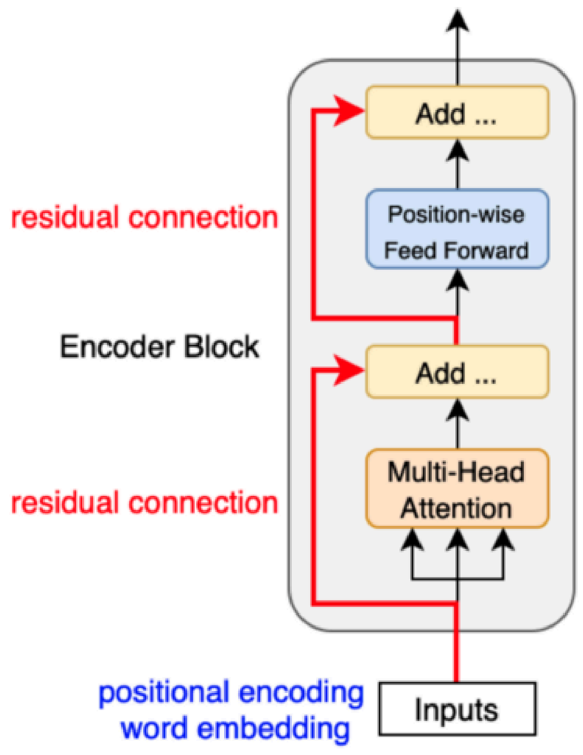
Encoder Block Internals

The encoder block utilizes the self-attention mechanism to enhance each token (embedding vector) with contextual information from the entire sentence.
Due to the potential variety of semantics and functions surrounding each token, the self-attention mechanism employs multiple heads (eight parallel attention calculations). This allows the model to access different embedding subspaces. For further insights into the self-attention mechanism, please refer to this article.

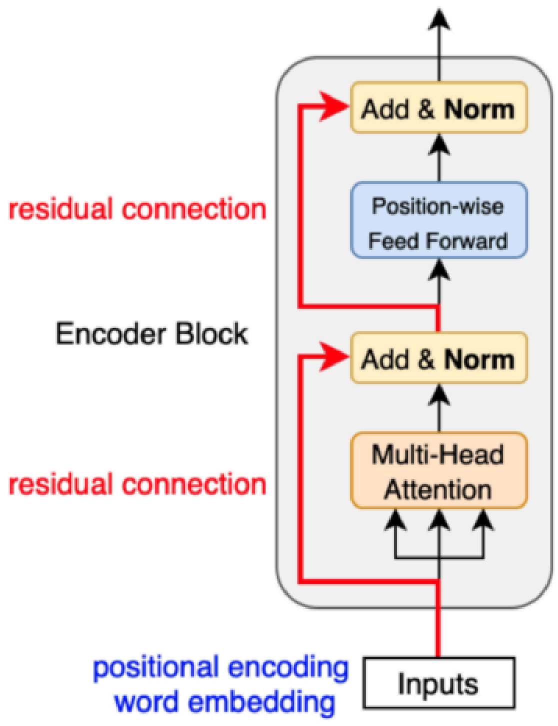


The position-wise feed-forward network (FFN) consists of a linear layer, followed by a ReLU activation function, and another linear layer. This network operates independently on each embedding vector, utilizing identical weights for processing. Therefore, each embedding vector, enriched with contextual information from the multi-head attention mechanism, undergoes further transformation through the position-wise feed-forward layer.

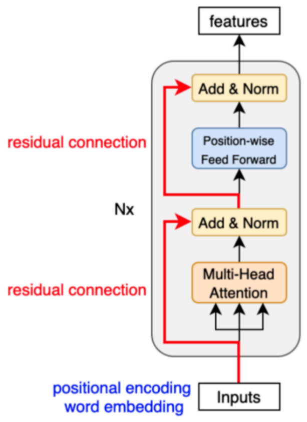
Another key aspect is that the encoder block employs residual connections, which involve element-wise addition:
Note: Sublayer is either multi-head attention or point-wise feed-forward network.



Residual connections propagate the previous embeddings to subsequent layers. Consequently, the encoder blocks augment the embedding vectors with extra information derived from the multi-head self-attention calculations and position-wise feed-forward networks.
Following each residual connection, there is a layer normalization step:



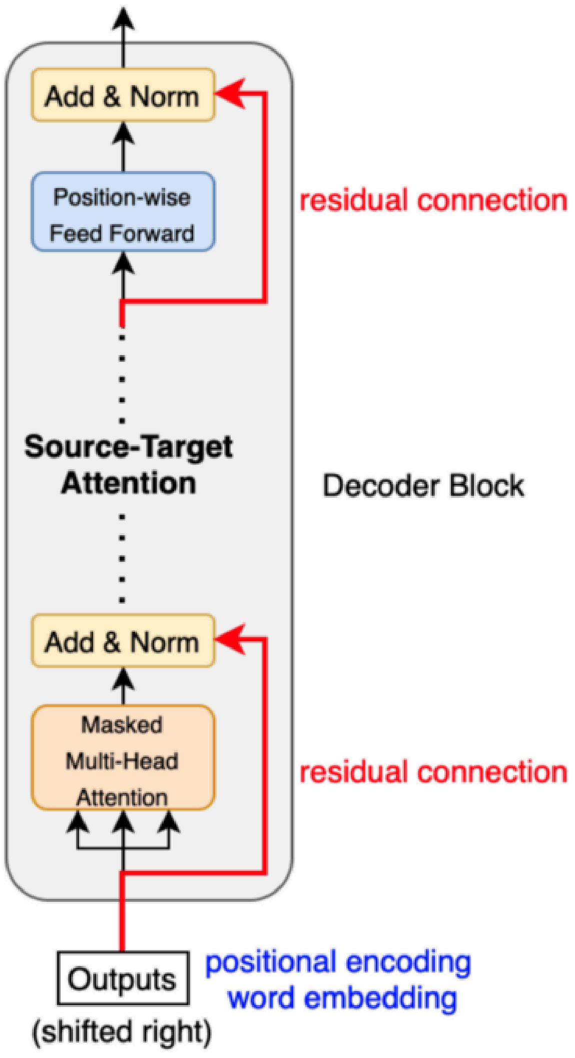
Layer normalization, akin to batch normalization, addresses covariant shift, thereby stabilizing and accelerating training by maintaining the mean and standard deviation of embedding vector elements. Unlike batch normalization, layer normalization operates on each embedding vector individually, rather than at the batch level. Layer normalization was initially introduced by Geoffrey Hinton's team as applying batch normalization to recurrent neural networks proved impractical.
The transformer comprises six stacked encoder blocks. The outputs from the final encoder block serve as the input features for the decoder.



As observed, the input features consist of a sequence of enriched embeddings obtained through multi-head attention mechanisms and position-wise feed-forward networks, incorporating residual connections and layer normalizations.
Now, let's examine how the decoder utilizes these input features.

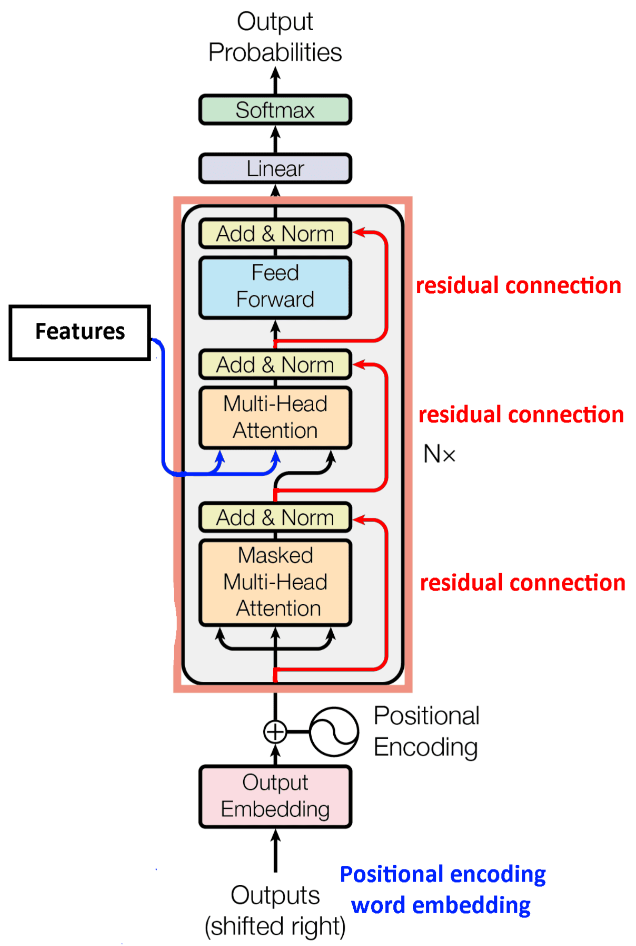
Decoder Block Internals

The decoder block closely resembles the encoder block, with the key distinction being that it computes the source-target attention.

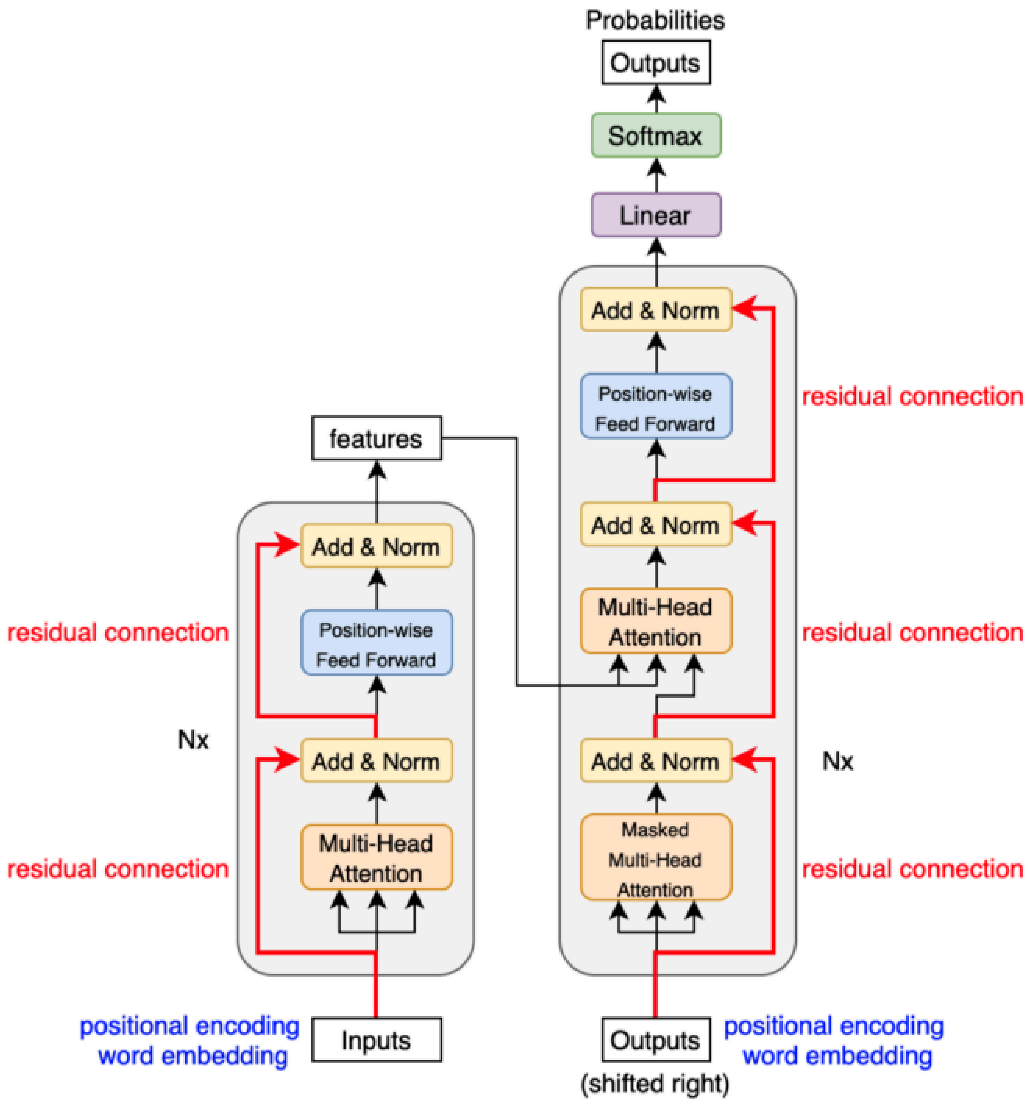


As previously mentioned, an input to the decoder is a right-shifted output, resulting in a sequence of embeddings with positional encoding. Consequently, we can conceptualize the decoder block as another encoder, generating enriched embeddings pertinent to translation outputs.
Masked multi-head attention entails providing inputs with masks to the attention mechanism, preventing it from utilizing information from hidden (masked) positions. The paper specifies that they applied the mask within the attention calculation by assigning negative infinity (or a very large negative number) to attention scores. The softmax function within the attention mechanisms effectively assigns zero probability to masked positions.
In essence, this process is akin to gradually increasing the visibility of input sentences through the masks:
(1, 0, 0, 0, 0, …, 0) => (<SOS>)
(1, 1, 0, 0, 0, …, 0) => (<SOS>, ‘Bonjour’)
(1, 1, 1, 0, 0, …, 0) => (<SOS>, ‘Bonjour’, ‘le’)
(1, 1, 1, 1, 0, …, 0) => (<SOS>, ‘Bonjour’, ‘le’, ‘monde’)
(1, 1, 1, 1, 1, …, 0) => (<SOS>, ‘Bonjour’, ‘le’, ‘monde’, ‘!’)

The source-target attention is another form of multi-head attention that computes attention values between the features (embeddings) from the input sentence and the features from the output (albeit partial) sentence.



Therefore, the decoder block enhances the embeddings by incorporating features from both the input and partial output sentences.



The full transformer architecture via KiKaBen

BLIP (Wip)

(Bootstrapping Language-Image Pre-training) for Unified Vision-Language Understanding and Generation

BLIP Overview:

- Bootstrapped Language-Image Pre-training (BLIP) combines text and image information during pre-training.
- BLIP bootstraps a pre-trained language model (such as BERT or RoBERTa) by adding a visual pathway. The visual pathway processes image features and aligns them with text tokens. (Pretraining on 3 web based ~14M, 2 human annotated ~450K, additional noisy web based ~115M and Finetuning on COCO & more)
- BLIP enables better understanding of context and improves performance on multimodal tasks.
- A Bootstrapping method for the data (CapFilt), including training captioners and filters which means that there's part that synthetically generate data and there is the part learns to distinguish good from bad data and that allows them to collect lots of data from the Internet and filtered out poorly labeled images and allows them to augment the dataset by labeling images themselves

Author's prior work ALing BEfore Fuse (ALBEF)

Review the three losses that will be used in the BLIP paper:

1.Image-text contrastive loss (ITC): This loss function encourages the model to learn similar representations for image and text that describe the same content, while pushing dissimilar representations further apart. It helps the model understand the relationship between visual and textual information.

It learns a **similarity function s** such that **parallel image-text pairs have higher similarity scores:**

$$s = g_v(v_{\text{cls}})^\top g_w(w_{\text{cls}}),$$

Where ***gv*** and ***gw*** are linear transformations that map the [CLS] embedding to normalized lower-dimensional (256-d) representation.Taken inspiration by MoCo: Momentum Contrast for Unsupervised Visual Representation Learning, two queues are maintained to store the most recent M image-text representation ***g'v, g'w***, from the momentum unimodal encoders, for calculating the similarity score:

$$s(I,T) = g_v(v_{\text{cls}})^\top g'_w(w'_{\text{cls}}) \text{ and } s(T,I) = g_w(w_{\text{cls}})^\top g'_v(v'_{\text{cls}}).$$

For each image and text, the softmax-normalized image-to-text and text-to-image similarities are calculated as:

$$p_m^{\mathrm{i2t}}(I) = \frac{\exp(s(I, T_m)/\tau)}{\sum_{m=1}^{M} \exp(s(I, T_m)/\tau)}, \quad p_m^{\mathrm{t2i}}(T) = \frac{\exp(s(T, I_m)/\tau)}{\sum_{m=1}^{M} \exp(s(T, I_m)/\tau)}$$

Where $\boldsymbol{\tau}$ is a learnable **temperature** parameter. Now let $yi2t(I)$ and $yt2i(T)$ denote the ground-truth one-hot similarity, where negative pairs have a probability of 0 and the positive pair has a probability of 1. The ITC is defined as the cross entropy of H between p and y:

$$\mathcal{L}_{\mathrm{itc}} = \frac{1}{2}\mathbb{E}_{(I,T)\sim D}\left[\mathrm{H}(\boldsymbol{y}^{\mathrm{i2t}}(I), \boldsymbol{p}^{\mathrm{i2t}}(I)) + \mathrm{H}(\boldsymbol{y}^{\mathrm{t2i}}(T), \boldsymbol{p}^{\mathrm{t2i}}(T))\right]$$

2.Masked language modeling loss (MLM): This loss focuses on classifying whether an image-text pair corresponds to each other. It strengthens the model's ability to identify matching visual and textual descriptions to predict masked words. Similar to BERT, the input tokens are randomly masked out with a probability of 15% and replace them with a special token [MASK].

$$\mathcal{L}_{\mathrm{mlm}} = \mathbb{E}_{(I,\hat{T})\sim D}\mathrm{H}(\boldsymbol{y}^{\mathrm{msk}}, \boldsymbol{p}^{\mathrm{msk}}(I, \hat{T}))$$

3.Image-text matching loss (ITM) with momentum distillation: This loss function is commonly used in language models like BERT. It helps BLIP improve its general understanding of language structure and improve the quality of generated text descriptions.

ITM predicts whether a pair of image and text is positive (matched) or negative (not matched). The multimodal encoder’s output embedding of the [CLS] token is used as the joint representation of the image-textpair, and **fully-connected (FC) layer is appended** and **followed by softmax** to **predict a two-class probability pitm**.

$$\mathcal{L}_{\mathrm{itm}} = \mathbb{E}_{(I,T)\sim D}\mathrm{H}(\boldsymbol{y}^{\mathrm{itm}}, \boldsymbol{p}^{\mathrm{itm}}(I, T))$$

The total loss is:

$$\mathcal{L} = \mathcal{L}_{\text{itc}} + \mathcal{L}_{\text{mlm}} + \mathcal{L}_{\text{itm}}$$

Paper Overview:
Junnan Li Dongxu Li Caiming Xiong Steven Hoi
The model employs Vision Transformer (ViT), which segments the input image into patches and encodes them as a sequence of embeddings. Additionally, it adds a [CLS] token to represent the global image feature. According to the authors, ViT offers lower computational costs and simplicity, making it increasingly adopted by recent methods.

To enable training/pre-training of such a model for understanding and generation tasks, the authors proposed a multimodal Mixture of Encoder and Decoder that integrates three functionalities, as depicted here:

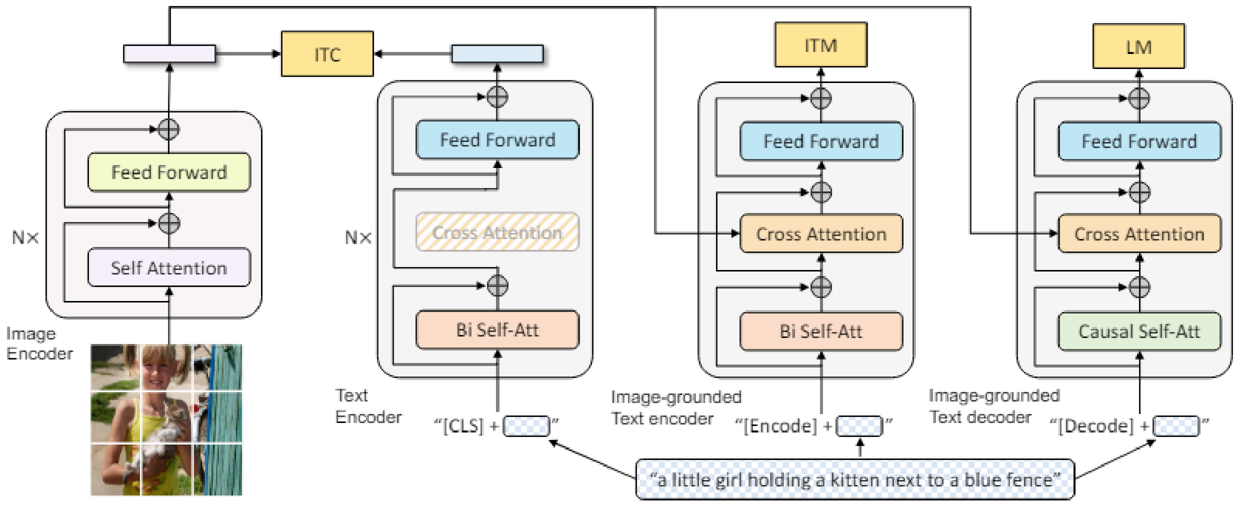


Figure 1. BLIP is a combination of existing things so an arrangement of modules for multi-task pre-training. This model will take in an image text pair and perform multiple tasks on it and has multiple losses and therefore ends up being able to do multiple things

A Text Encoder (BERT) and Image Encoder(VIT) on the left, an Image grounded Text Encoder on the middle and a Decoder on the right

The proposed multimodal mixture of encoder-decoder has three functionalities:

1. Text Encoder and Image Encoder (Unimodal encoder): Trained with an image-text contrastive (ITC) approach.
2. Image-grounded text encoder: Utilizes additional cross-attention layers to capture vision-language interactions and is trained with an image-text matching (ITM) loss.
3. Image-grounded text decoder: Substitutes the bi-directional self-attention layers with causal self-attention layers. It shares the same cross-attention layers and feedforward networks as the encoder.

Now let us discuss each block in more detail:

1) Unimodal encoder:

The unimodal encoder encodes both text and image, as illustrated in the figure below. The text encoder mirrors BERT, specifically the Masked Language Model (Devlin et al., 2019), incorporating a [CLS] token at the onset of the text input to summarize the sentence.

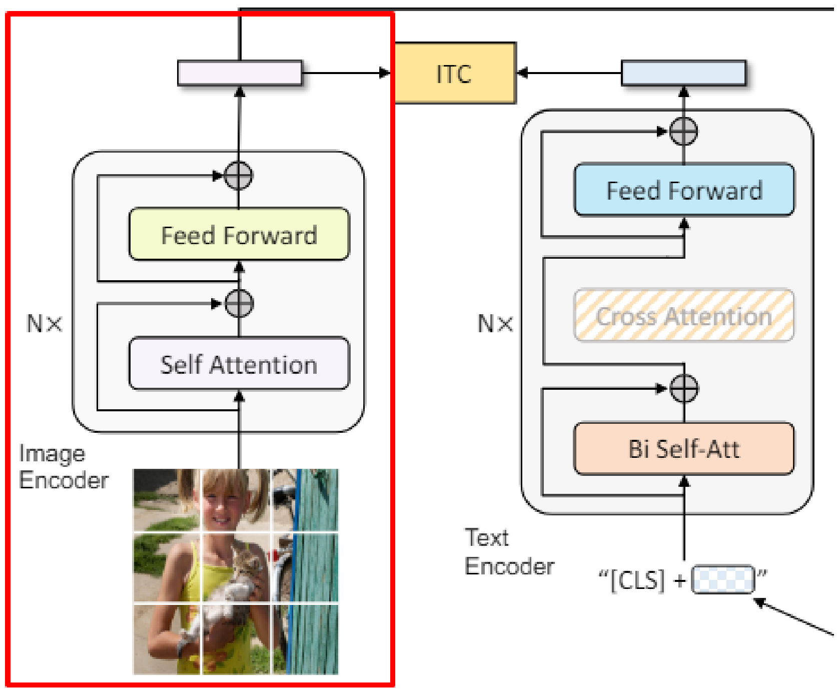
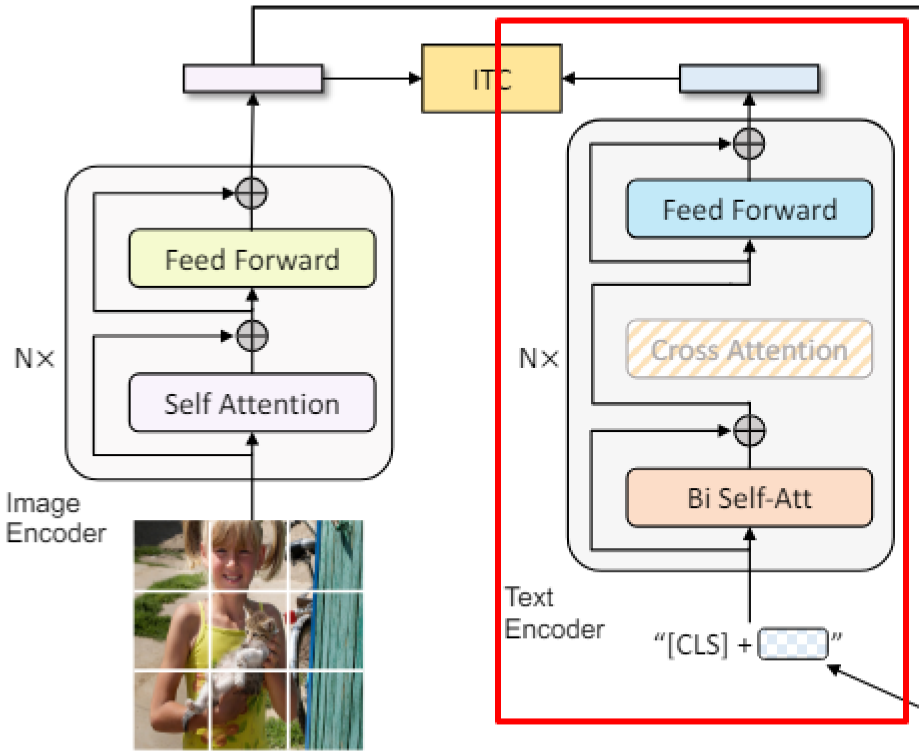


Figure 2. Unimodal encoder. The encoder is trained with an image-text contrastive (ITC) loss to align the vision and language representations.

The first component of the model is the vision Transformer Encoder which uses the same architecture from the paper VIT, we have an input image worth 16x16 words, during training each image is resized to a resolution of 224x224 and then during fine tuning they resized to 384x384. In the forward pass each image is divided into 16x16

patches with each patches flattened and stacked on top of each other and then an extra class token is prepended to the sample which represents the global image feature then an positional encodings are added to each patch and the embedding features are passed to the **Self Attention**.
Now using Key, Query and Value matrices the Self Attention scores for the samples are computed and are then added back to the original embedding with the skip connection and then normalized across the layer. These normalized attention scores are then passed through a **Feed Forward** Layer and then again are added to the original attention scores and normalized across the layer.
The Output of the Image Encoder is linearly projected and will be used for all other components.



Next up we have the Text Transformer which as mentioned used the pre-trained BERT based model to encode the text samples. For each forward pass the tokens are embedded and like the VIT a class token is prepended to the sequence. The [CLS] token represents the summarization of the sentence.
Unlike the Image Encoder the Text encoder Self Attention Layer is changed with a Bidirectional self attention similar to the BERT model. Which means that in this Bi Self-Att 15% of tokens in a sequence are masked and of this 15%, 80% of the tokens are substituted with the mask token ,10% are exchanged with a random token and then 10% of the tokens remain unchanged. By taking information from both right and left

sides of the tokens context, the Bi Self-Att generates improved token embedding as well as an improved feature representation for each text sample.
The Output of the Text Encoder is linearly projected to the vector of the same dimensionality of the Vision Transformer Output.
Now During the backward pass, the image text contrast of loss is computed using the outputs of both vision and text encoder, in order to align each modality feature representation in a global feature space.

ITC: $$\mathcal{L}_{\text{itc}} = \frac{1}{2}\mathbb{E}_{(I,T)\sim D}\left[\mathrm{H}(\boldsymbol{y}^{\text{i2t}}(I), \boldsymbol{p}^{\text{i2t}}(I)) + \mathrm{H}(\boldsymbol{y}^{\text{t2i}}(T), \boldsymbol{p}^{\text{t2i}}(T))\right]$$

The objective of contrastive pre-training is to jointly train an image and text encoder such that the cosine similarity of the correct image text pair is maximized. At the same time the cosine dissimilarity image-text pair should be minimized. This is accomplished by stacking all images in a batch on top of each other. Creating a matrix which is similarly done to the text samples. By multiplying the image matrix by the transpose of the text Matrix, a global feature space is computed where the diagonal of said matrix should represent intersection of each correct image text pair .

2) Image-grounded text encoder:

Like the previous text encoder each word token is embedded and a learnable task specific token is prepended to the sequence. Notice that this encoder incorporates an extra cross-attention (CA) layer, positioned between the self-attention (SA) layer and the feed-forward network (FFN) within each transformer block of the text encoder. This additional CA layer is task-specific,it takes the final query, key matrices generated from the vision Transformer and calculates the self attention scores using the value matrix of the image ground text encoder. This mechanism injects visual information into the attention scores of the image.

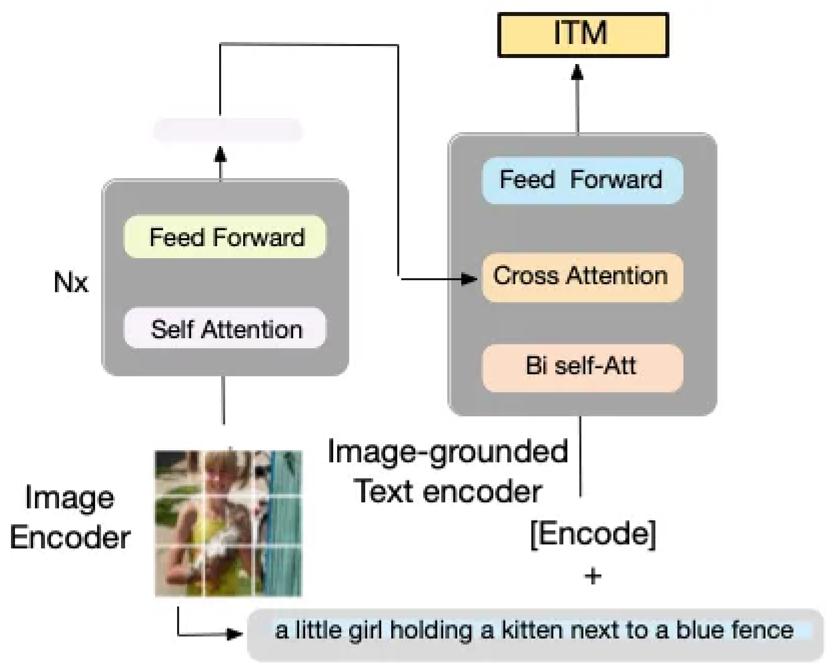


Figure 3. Image-grounded text encoder. The encoder uses additional cross-attention layers to model vision-language interactions, and is trained with an image-text matching (ITM) loss to distinguish between positive and negative image-text pairs.

The Image-grounded text encoder employs the Image-Text Matching Loss (ITM) to capture precise alignment between vision and language. ITM is structured as a binary classification task, where the model predicts whether pairs are positive matches or negative mismatches, essentially functioning as a filter. Notably, to enrich the dataset with negative pairs, a hard negative mining strategy is employed, utilizing contrastive similarity to identify the most similar negative pair (Li et al., 2021a).

3) Image-grounded text decoder:

Like the image-grounded text encoder, the textual tokens are embedded and decoded. An EOS token prepended to the sequence to represent the beginning of the sequence and the end of the sentence token respectively. They both share the embedding layer, the cross attention layers and the feedforward layers. This method was employed because all these layers are not specific for a particular task and it improves the computational and story efficiency of the model. However unlike the encoder the Image-ground text decoder replaces bidirectional self-attention with a causal self-attention mask. This masking technique confines Transformer queries to the preceding key value pair positions as well as the current key position.

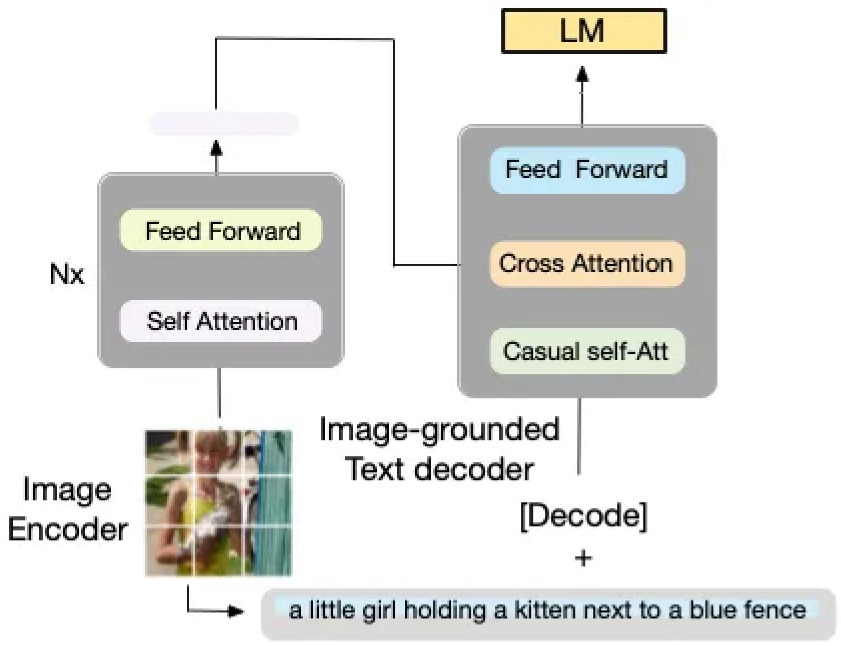


Figure 4. Image-grounded text decoder. The decoder replaces the bi-directional self-attention layers with causal self-attention layers, shares the same cross-attention layers, and feed-forward networks as the encoder. The decoder is trained with a language modeling (LM) loss to generate captions given images.

The Image-grounded text decoder utilizes Language Modeling Loss (LM), which is activated to generate textual descriptions based on an image input. The LM loss is trained to maximize the likelihood of the text in an autoregressive manner. This Masked Language Modeling loss has demonstrated significant effectiveness in various models, including visual and language pre-training, for generating coherent captions.

$$\mathcal{L}_{\text{itm}} = \mathbb{E}_{(I,T)\sim D}\mathrm{H}(y^{\text{itm}}, p^{\text{itm}}(I,T))$$

…

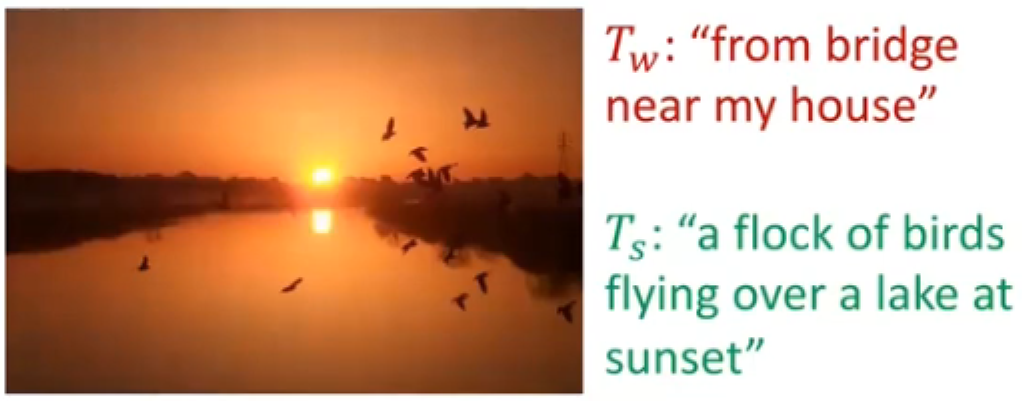
To facilitate efficient pre-training while leveraging multi-task learning, the following strategies are employed:

- Parameter Sharing: The text encoder and text decoder share all parameters except for the self-attention layers. This is illustrated in Figure 1, where the same color indicates shared parameters. The rationale behind this approach is that the distinctions between encoding and decoding tasks are most effectively captured by the self-attention layers.
- Self-Attention Layers: The encoder utilizes **bi-directional** self-attention to construct representations for the current input tokens, whereas the decoder

utilizes **causal** self-attention to predict the subsequent tokens. By sharing these layers, training efficiency is enhanced while simultaneously benefiting from multi-task learning.

CapFilt (Captioning and Filtering)

Due to the high cost associated with human annotation, there exists only a limited number of high-quality human-annotated image pairs, such as those found in COCO dataset (Lin et al., 2014). In contrast, most recent works (Li et al., 2021a) heavily rely on noisy data extracted from the web through web scraping (WS). This process offers a rapid means of gathering a larger volume of image-text pairs without requiring human involvement. However, since the data sourced from web scraping is noisy, containing incorrect descriptions of images, as illustrated in the image below, it adversely impacts the learning of vision-language alignment signals.


T_w: "from bridge near my house"
T_s: "a flock of birds flying over a lake at sunset"

A wrong image-par from the web (human written). **Caption:** from bridge near my house

To address this issue, the authors propose "Captioning and Filtering" (CapFilt), a novel approach designed to enhance the quality of the text corpus. Figure 5 provides an illustration of CapFilt, which incorporates two key modules:

1. Captioner: This module generates synthetic captions for web images.
2. Filter: This module is responsible for removing noisy image-text pairs.

The goal of CapFilt is to take noisy web data, filter out unusable captions, and provided higher quality synthetic captions

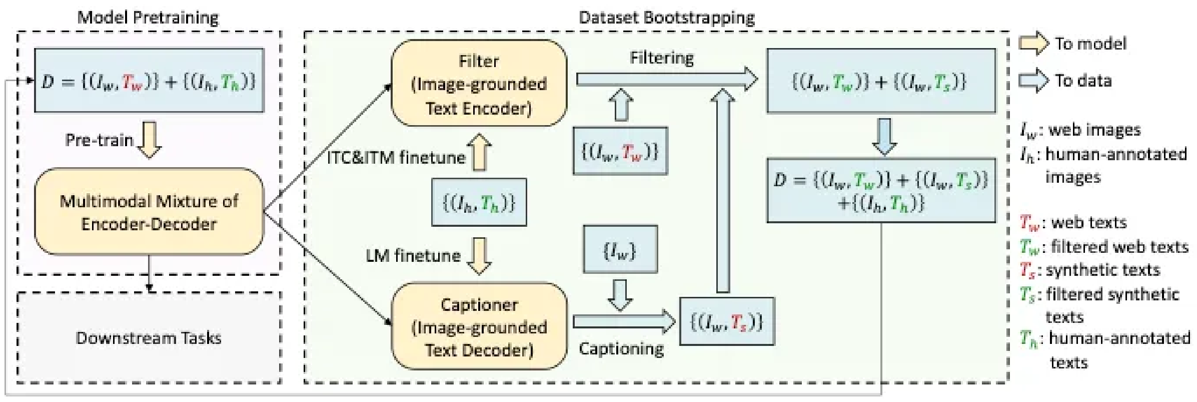

Model Pretraining
Dataset Bootstrapping
$D = \{(I_w, T_w)\} + \{(I_h, T_h)\}$
Pre-train
Multimodal Mixture of Encoder-Decoder
Downstream Tasks
Filter (Image-grounded Text Encoder)
Captioner (Image-grounded Text Decoder)
ITC&ITM finetune
LM finetune
$\{(I_h, T_h)\}$
Filtering
Captioning
$\{(I_w, T_w)\}$
$\{I_w\}$
$\{(I_w, T_s)\}$
$\{(I_w, T_w)\} + \{(I_w, T_s)\}$
$D = \{(I_w, T_w)\} + \{(I_w, T_s)\} + \{(I_h, T_h)\}$
To model
To data
I_w: web images
I_h: human-annotated images
T_w: web texts
T_w: filtered web texts
T_s: synthetic texts
T_s: filtered synthetic texts
T_h: human-annotated texts

Figure 5. The Learning framework of BLIP. A captioner is introduced via sampling to produce synthetic captions for web images, and a filter to remove noisy image-text pairs. The captioner and filter are initialized from the same pre-trained model and fine-tuned individually on a small-scale human-annotated dataset.

Based on Figure 5, the captioner is essentially an image-grounded text decoder, as depicted in Figure 4. Here's how the CapFilt method works:

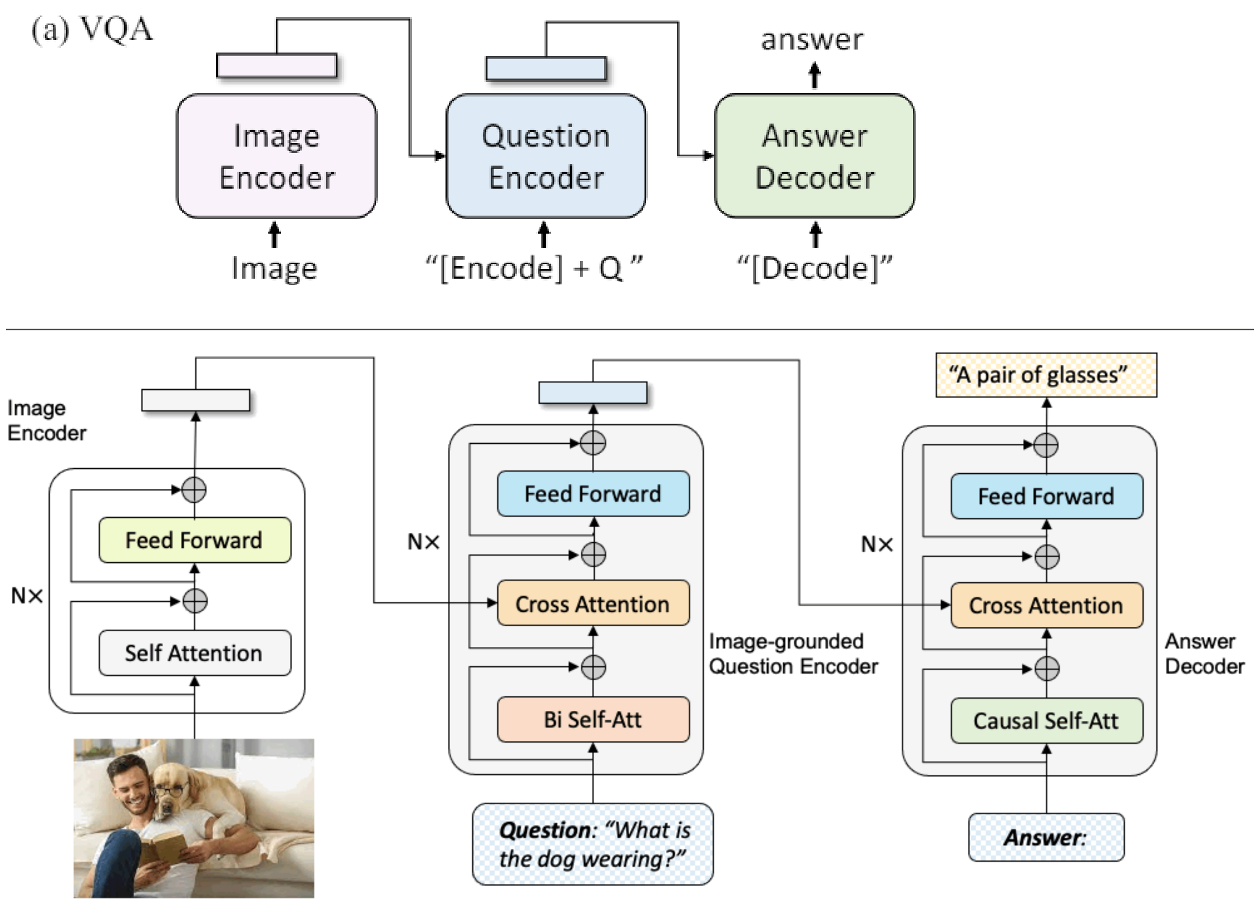
1. **Captioner:** Using the web images Iw , the captioner generates synthetic captions T_S, producing one caption per image. It achieves this by fine-tuning the LM objective to decode texts given images.
2. **Filter:** The filter, on the other hand, acts as an image-grounded text encoder. Fine-tuned with the ITC and ITM objectives, it learns to determine whether a text matches an image. The filter's role is to eliminate noisy texts from both the original web texts T_w and the synthetic texts T_S. A text is identified as noisy if the ITM head predicts it as unmatched to the image.
3. **Combination:** Finally, all the aforementioned steps are integrated, combining the filtered image-text pairs with the human-annotated pairs. This amalgamation forms a new dataset that will be utilized to pre-train a new model.

Effect of CapFilt

The introduction of CapFilt, which involves captioning and filtering noisy data from the web, presents a significant opportunity to scale models with higher accuracy due to reduced noise within the image-text pairs. By leveraging CapFilt to filter out noisy web data, notable performance improvements are observed across various downstream tasks, including image-text retrieval and image captioning, in both fine-tuned and zero-shot settings.

Moreover, the authors note that applying either the captioner or the filter alone to a dataset containing 14 million images results in observable performance enhancements. This underscores the effectiveness of each component individually in **improving** model performance.

BLIP on VQA (Visual Question Answering)



Note:
Parameter Sharing and Decoupling During pre-training. The text encoder and decoder share all parameters except for the self-attention layers as shown in the main Figure 1

May also check out "BLIP-2: Bootstrapping Vision-Language Pre-training with Frozen Unimodal Models", presents a novel approach to vision-language pre-training. It leverages the strengths of pre-trained unimodal models and introduces a Querying Transformer (Q-Former) to bridge the gap between vision and language modalities. This article provides a detailed explanation of the BLIP-2 architecture, its training stages, and the inference process.

Dataset Construction

1) Dataset Description

Overview

The PathVQA dataset is specifically designed for training and testing Medical Visual Question Answering (VQA) systems in English. It contains question-answer pairs related to pathology images. Let's delve into the key details:

- **Purpose:** The dataset serves as a valuable resource for developing and evaluating VQA models in the medical domain.
- **Question Types:**
  - Open-ended questions: These allow for diverse answers.
  - Binary "yes/no" questions: These have a binary answer format.
- **Answer Types:**
  - Some datasets categorize answers (e.g., "yes/no," "number," "color").
- **Question-Answer Pairs:**
  - Each image corresponds to one or more questions.
  - Multiple answers per question due to human ambiguity.
- **Source:**
  - The dataset is constructed from two publicly-available pathology textbooks which are "Textbook of Pathology" and "Basic Pathology"
  - Additionally, content from the publicly-available digital library, "Pathology Education Informational Resource" (PEIR), contributes to the dataset.
- **Copyrights:**
  - Licensing Information:The authors have released the dataset under the MIT License
  - Citation Information:

```
@article{he2020pathvqa,
   title={PathVQA: 30000+ Questions for Medical Visual Question Answering},
   author={He, Xuehai and Zhang, Yichen and Mou, Luntian and Xing, Eric and Xie,
Pengtao},
   journal={arXiv preprint arXiv:2003.10286},
   year={2020}
}
```

2) Dataset Summary

Version and Size

- **Version:** The dataset version used is from an updated Google Drive link shared by the authors on February 15, 2023
- **Image and Question-Answer Pair Statistics:**
    - Total images: 5,004
    - Question-answer pairs: 32,795
    - Referenced images (used in pairs): 4,289
    - Unused images: 715
- **Duplicate Removal:**
    - Some image-question-answer triplets occur more than once within the same split (training, validation, test).
    - Duplicate triplets were meticulously removed.
- **Final Dataset:**
    - After handling duplicates, the dataset contains 32,632 question-answer pairs associated with 4,289 unique images.
- **Data Fields:**
    - “Image”: the image referenced by the question-answer pair.
    - “Question”: the question about the image.
    - “Answer”: the expected answer
- **Data Splits:** The dataset is split into training, validation and test sets provided directly by the author which ensure balanced representation of question types and image content across splits

3) Supported Tasks and LeaderBoards

Evaluation Metrics

- The PathVQA dataset has an active leaderboard on Papers with Code.
- Models are ranked based on three metrics:
    - Yes/No Accuracy: Accuracy for binary “yes/no” questions.
    - Free-form Accuracy: Accuracy for open-ended questions.
    - Overall Accuracy: Accuracy across all question types.

Model Training

1. Introduction:

In this section, we describe the process of training a VQA model using the Bootstrapped Language-Image Pre-training (BLIP) approach on the PathVQA dataset. Our goal is to develop an effective VQA system capable of answering questions related to medical images.

2. Dataset Preparation:

PathVQA Dataset

- Description:
    - The PathVQA dataset contains question-answer pairs related to pathology images.
    - It includes both open-ended questions and binary “yes/no” questions.
    - Sourced from publicly-available pathology textbooks and a digital library.
- Size:
    - Total images: 5,004
    - Question-answer pairs: 32,795
- Preprocessing:
    - Resize images to a consistent resolution (e.g., 224x224 pixels).
    - Normalize pixel values.
    - Tokenize and clean textual questions.
    - Create a vocabulary for question words.

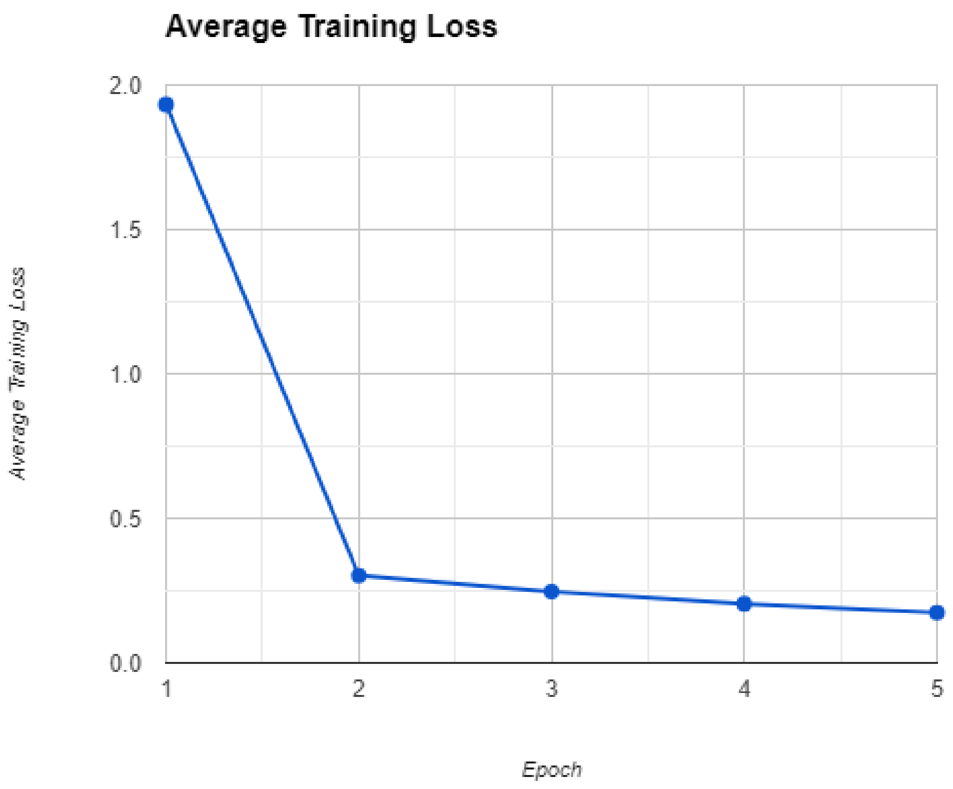
3. Model Architecture and initialization:

- We use a pre-trained BLIP model specifically designed for VQA tasks using the BlipForQuestionAnswering from the transformers library
- We initialize the model with this: BlipForQuestionAnswering.from_pretrained("Salesforce/blip-vqa-base")

4. Training Procedure

After initializing our model we begin to fine-tune the BLIP model on the PathVQA dataset on the Medical domain

- Set the learning rate at 0.00005
- Use a suitable optimizer which is AdamW
- Train the model on 5 epochs (machine crash at 8 epoch T-T)

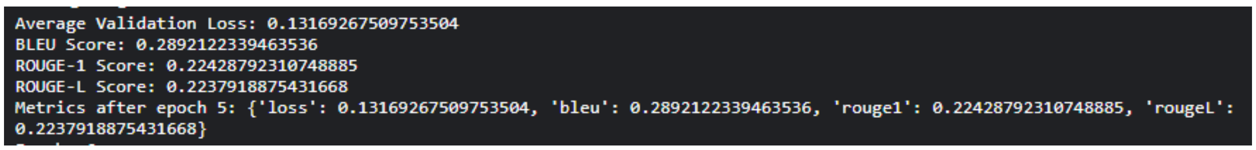


Testing and evaluate accuracy

1. Evaluation Metrics
We assess the performance of our VQA model using the following metrics:

- Average Validation Loss: Measures the model's performance during validation.
- BLEU Score: Evaluates the quality of generated answers by comparing them to reference answers. It considers n-grams overlap.
- ROUGE-1 Score: Computes the overlap of unigram (single-word) tokens between generated and reference answers.
- ROUGE-L Score: Considers the longest common subsequence between generated and reference answers.

2. Results



```
Average Validation Loss: 0.13169267509753504
BLEU Score: 0.2892122339463536
ROUGE-1 Score: 0.22428792310748885
ROUGE-L Score: 0.2237918875431668
Metrics after epoch 5: {'loss': 0.13169267509753504, 'bleu': 0.2892122339463536, 'rouge1': 0.22428792310748885, 'rougeL': 0.2237918875431668}
```

3. Interpretation
   - Lower validation loss indicates better model performance.
   - Higher BLEU and ROUGE scores signify better alignment with reference answers.
4. Conclusion
   - Our VQA model achieves promising results based on the evaluation metrics.
   - Further analysis and fine-tuning can enhance its accuracy.

References:

Lecture 10-BLIP:Bootstrapping Language-Image Pretraining for Unified VL Understanding…
https://visualqa.org/
Dataset:
https://huggingface.co/datasets/flaviagiammarino/path-vqa
Blip:
https://arxiv.org/pdf/2201.12086
https://arxiv.org/abs/2107.07651 (Align before fuse)
https://sh-tsang.medium.com/brief-review-align-before-fuse-vision-and-language-representation-learning-with-momentum-34386305ce0c
https://ahmed-sabir.medium.com/paper-summary-blip-bootstrapping-language-image-pre-training-for-unified-vision-language-c1df6f6c9166
UNIMO
Nucleus sampling
https://machinelearningmastery.com/what-is-attention/

https://machinelearningmastery.com/the-attention-mechanism-from-scratch/

https://machinelearningmastery.com/the-transformer-attention-mechanism/

https://machinelearningmastery.com/the-transformer-model/

https://huggingface.co/models

https://d2l.ai/chapter_attention-mechanisms-and-transformers/index.html

https://discuss.huggingface.co/t/continual-pre-training-vs-fine-tuning-a-language-model-with-mlm/8529

https://huggingface.co/docs/transformers/en/index

https://huggingface.co/models

https://heidloff.net/article/foundation-models-transformers-bert-and-gpt/

https://kikaben.com/transformers-encoder-decoder/
Pretraining:

https://www.philschmid.de/pre-training-bert-habana

Inference:

https://huggingface.co/blog/bert-inferentia-sagemaker

How a Transformer works at inference vs training time

3Blue1Brown:

https://www.youtube.com/watch?v=wjZofJX0v4M&t=57s

https://arxiv.org/pdf/2003.00744

https://arxiv.org/pdf/1907.11692

https://github.com/rohan-paul/LLM--Large-Language-Models/blob/main/Other-Language_Models_BERT_related/FineTuning_BERT_for_Multi_Class_Classification_Turkish/Multi-class_Classification.ipynb

Fine tuning LLMs: (Extras)

https://www.youtube.com/watch?v=XpoKB3usmKc

https://arxiv.org/pdf/2305.14314

https://platform.openai.com/docs/guides/fine-tuning/when-to-use-fine-tuning

Part 1 References:
https://machinelearningmastery.com/what-is-attention/

https://machinelearningmastery.com/the-attention-mechanism-from-scratch/

https://machinelearningmastery.com/the-transformer-attention-mechanism/

https://machinelearningmastery.com/the-transformer-model/

https://huggingface.co/models

https://d2l.ai/chapter_attention-mechanisms-and-transformers/index.html

https://discuss.huggingface.co/t/continual-pre-training-vs-fine-tuning-a-language-model-with-mlm/8529

https://huggingface.co/docs/transformers/en/index

https://huggingface.co/models

https://heidloff.net/article/foundation-models-transformers-bert-and-gpt/

https://kikaben.com/transformers-encoder-decoder/
Pretraining:

https://www.philschmid.de/pre-training-bert-habana

Inference:

https://huggingface.co/blog/bert-inferentia-sagemaker

How a Transformer works at inference vs training time

3Blue1Brown:

https://www.youtube.com/watch?v=wjZofJX0v4M&t=57s

https://arxiv.org/pdf/2003.00744

https://arxiv.org/pdf/1907.11692

https://github.com/rohan-paul/LLM--Large-Language-Models/blob/main/Other-Language_Models_BERT_related/FineTuning_BERT_for_Multi_Class_Classification_Turkish/Multi-class_Classification.ipynb

Fine tuning LLMs: (Extras)

https://www.youtube.com/watch?v=XpoKB3usmKc

https://arxiv.org/pdf/2305.14314

https://platform.openai.com/docs/guides/fine-tuning/when-to-use-fine-tuning