Attention Is All You Need

Research Report for the NLP Cryptonite Taskphase

By Pratham Shah, Research AI JTP 240905614

- Authors: Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin (Google Brain)
- Published: NeurIPS 2017
- PDF: Attention Is All You Need (NeurIPS PDF)

This report reviews the landmark paper “Attention Is All You Need” – a breakthrough for sequence-to-sequence tasks in NLP, with its invention of the Transformer architecture.

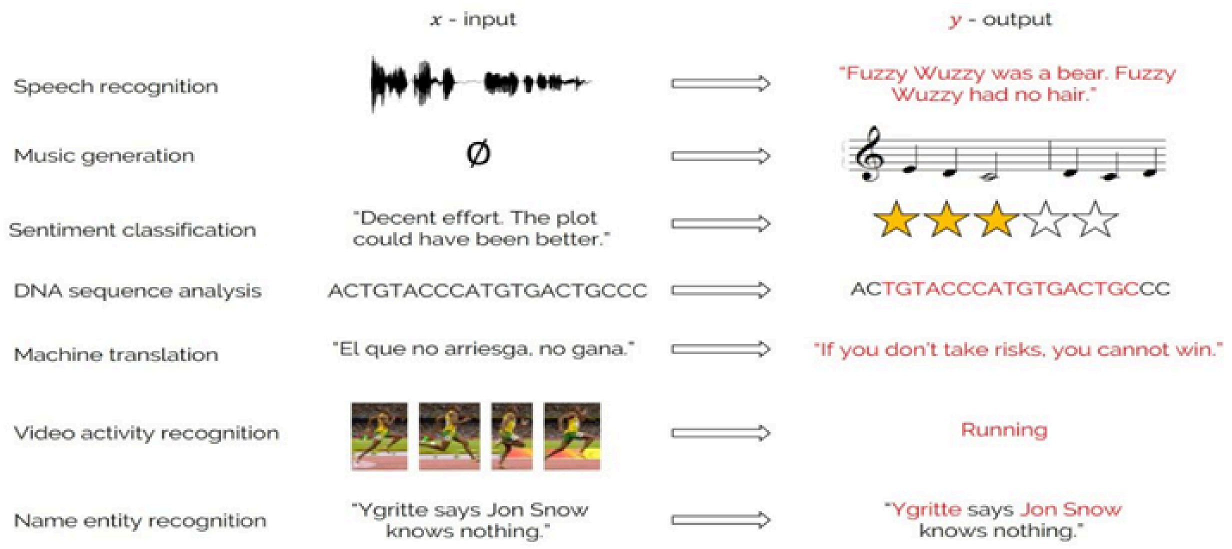
The study highlights how the Transformer eliminates previous recurrent and convolutional structures by relying entirely on self-attention mechanisms, enabling faster and more efficient parallel processing of sequences.

Key innovations like multi-head self-attention and positional encoding are discussed, along with comparative performance benchmarks that demonstrate state-of-the-art results in machine translation and set the foundation for modern language models.

Introduction

Sequence modeling is a core task in machine learning and AI, focused on learning from sequentially ordered data such as text, speech, time series, or biological sequences. The main challenge is to model dependencies and contextual patterns across the elements of a sequence, enabling predictions, labeling, generation, and understanding of long-range relationships.

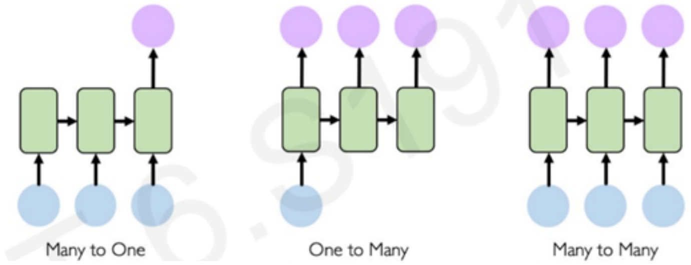
Sequence Modelling can predict future elements, classify labels, generate new sequences, and even capture long-range relationships in data.


x - input
y - output
Speech recognition
"Fuzzy Wuzzy was a bear. Fuzzy Wuzzy had no hair."
Music generation
∅
Sentiment classification
"Decent effort. The plot could have been better."
DNA sequence analysis
ACTGTACCCATGTGACTGCCC
ACTGTACCCATGTGACTGCCC
Machine translation
"El que no arriesga, no gana."
"If you don't take risks, you cannot win."
Video activity recognition
Running
Name entity recognition
"Ygritte says Jon Snow knows nothing."
"Ygritte says Jon Snow knows nothing."

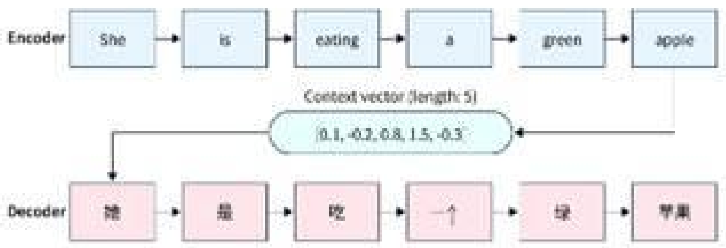
Sequence Modelling is an important aspect of Natural Language. Sentences are a sequential string of interdependent words. Each word changes based on the others, and a fixed set of rules govern their order and conjugation.

In natural language, sequence modeling is vital because the meaning of each word is sequentially context-dependent and governed by grammatical rules. Up to the 2010s, techniques like Hidden Markov Models (HMMs), Recurrent Neural Networks (RNNs), and Long Short-Term Memory networks (LSTMs) dominated the field but faced several limitations:

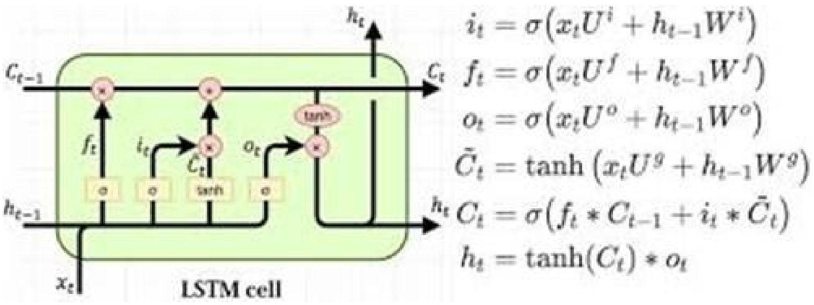
- **Lack of Parallelism:** Training was slow and sequential, as RNNs require each step's output before processing the next.

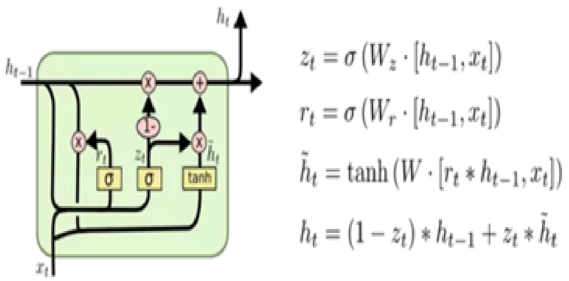


- **Bottleneck Representation:** Encoders pass a fixed-length hidden state to the decoder, often causing crucial old sequence information to be lost.



- **Gradient Issues:** Because of their length and recurrence, models encountered vanishing gradients (where learning "fades" as you go backward in the network) and exploding gradients (where learning becomes unstable with very large updates).


C_{t-1}
h_{t-1}
x_t
h_t
C_t
f_t
i_t
$\tilde{C}_t$
o_t
tanh
σ
LSTM cell
$i_t = \sigma(x_t U^i + h_{t-1} W^i)$
$f_t = \sigma(x_t U^f + h_{t-1} W^f)$
$o_t = \sigma(x_t U^o + h_{t-1} W^o)$
$\tilde{C}_t = \tanh(x_t U^g + h_{t-1} W^g)$
$C_t = \sigma(f_t * C_{t-1} + i_t * \tilde{C}_t)$
$h_t = \tanh(C_t) * o_t$


h_{t-1}
h_t
x_t
r_t
z_t
$\tilde{h}_t$
1-
σ
tanh
$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$
$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$
$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$
$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$

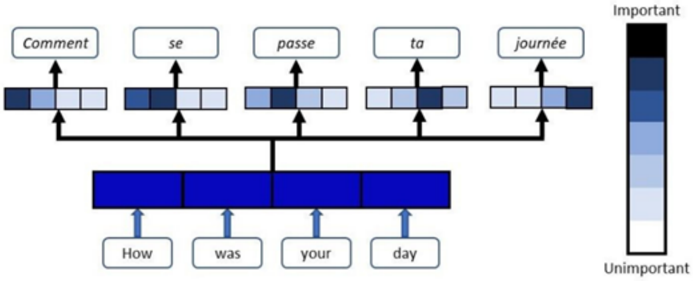
LSTMs and GRUs introduced mechanisms to better capture long-distance dependencies, but they still struggled with parallelism and scaling to long sequences, despite factorization and conditional computation tricks. CNNs had some parallelisation but the number of operations to relate signals increased with their distance – resulting in low retention again.

Problem Statement

Traditional models for sequence tasks in NLP, especially recurrent architectures, struggled to efficiently model long sequences and scale with data size. The challenges were:

- Inefficient training and inference due to unavoidable sequential processing.
- Inability to fully exploit hardware (especially GPUs) for parallel computation.
- Limited memory of long-range dependencies due to bottleneck states and gradient issues.

These challenges motivated the attention mechanism (Bahdanau et al.) which modeled dependencies irrespective of distance, and paved the way for the transformer by Vaswani et al, which removed all reccurence and put attention front-and-center.

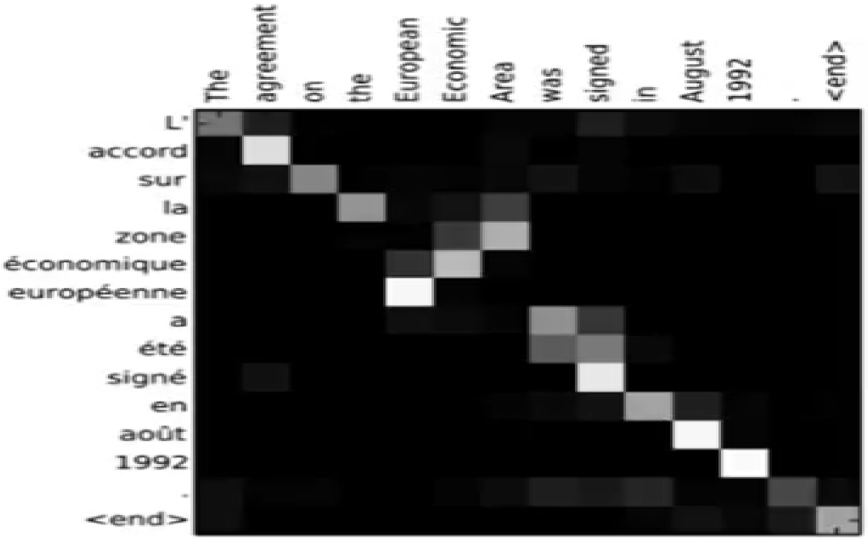


Attention Is All You Need: The Transformer’s Encoder-Decoder Structure

Attention is a mechanism in ML/NLP that makes models dynamically prioritize and focus on the most relevant segments of input data, when producing an output.

Instead of treating all parts of a sequence equally, the attention mechanism computes a set of weights that signal which tokens (words, etc.) should receive more emphasis at a given step.

This idea is inspired by how humans selectively focus on certain aspects of information when understanding language or images.

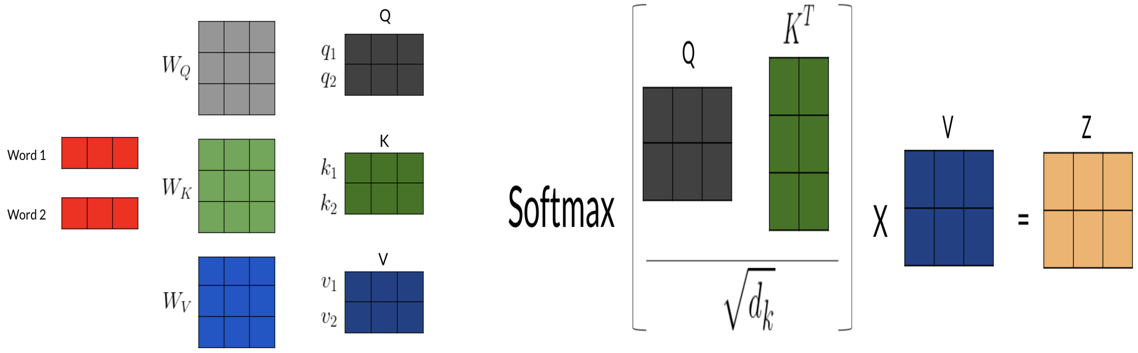


Example of How Attention Captures Related Words in a Sequence

Words are first represented as tokens, which are then mapped to their corresponding word embeddings (eg. Word2Vec). However, these embeddings are static — they do not account for context. For example, the word *"bank"* would have the same embedding whether it's used in the context of a river or a financial institution.

Self-attention transforms these static embeddings into context-aware representations, allowing the model to capture the nuanced meaning of a word based on the words around it — staying true to the idea that *"you shall know a word by the company it keeps."*

Assume we have a sequence of 2 word vectors (3 dimensional + added positional embeddings, in red), here's how we can give them additional context about their surroundings in the ***ENCODER:***

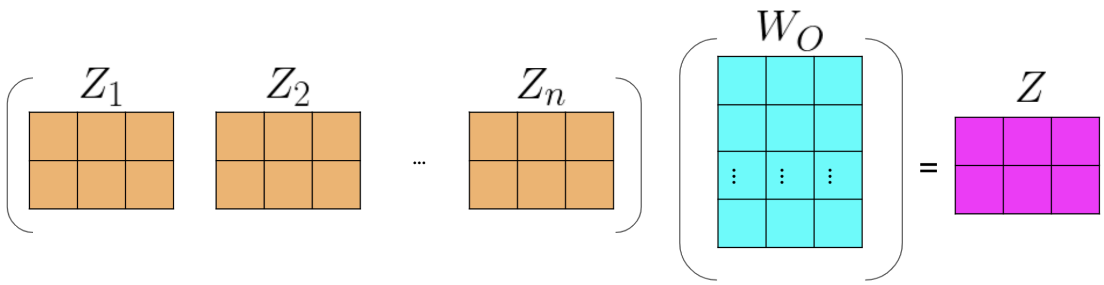


The WQ (Query), WK (Key), and WV (Value) matrices are learnable parameter matrices. Each is a square matrix the size of the embedding dim, typically initialized randomly and trained via backpropagation. Their roles are:

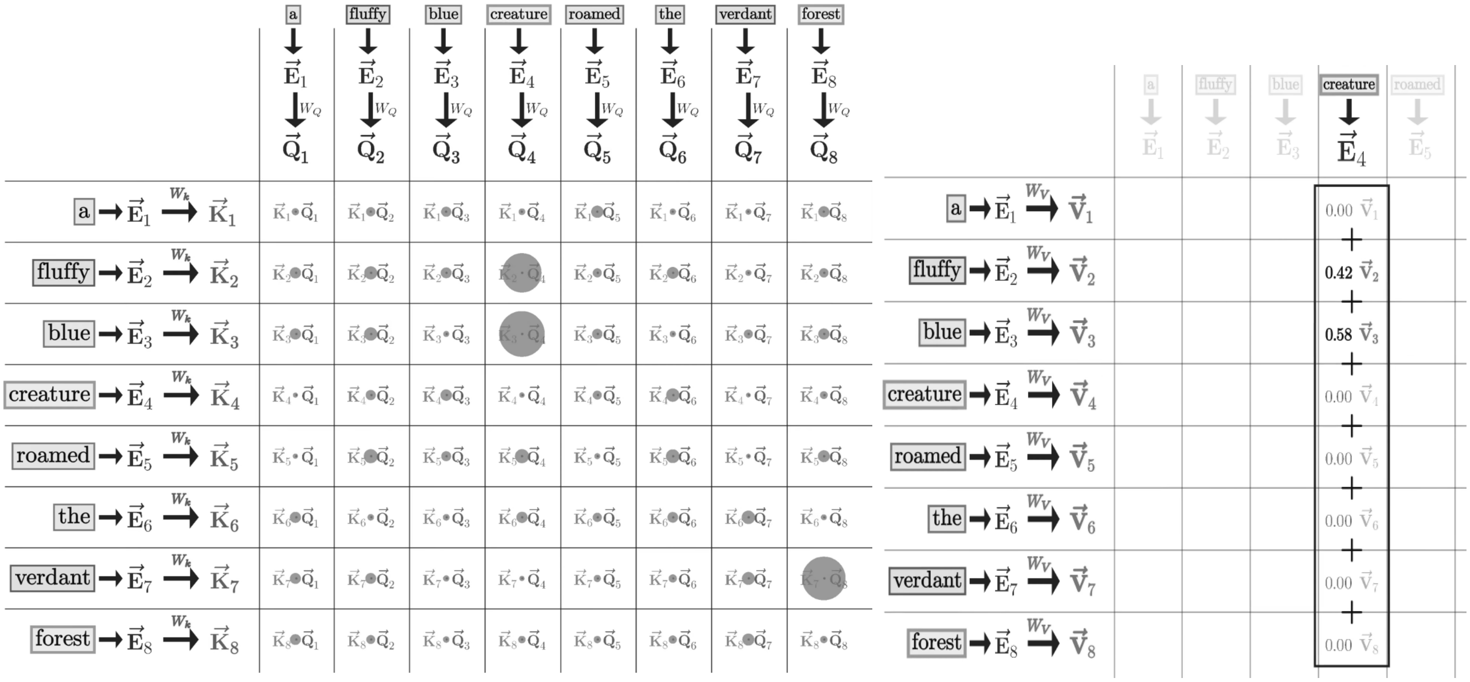
- WQ: Projects each input embedding into a query vector, representing "what this position is looking for" in other tokens.
- WK: Projects each input embedding into a key vector, encoding "what this token offers" to other queries.
- WV: Projects each input embedding into a value vector, containing the information to be aggregated as context.

Through these projections, self-attention can compute attention weights (via Q·K) and create new, context-aware representations (softmax weighted sums of Vs added to the original embedding) for each token in the input sequence.

In the case of multi headed attention, the Q, K and V matrices are split into d_embedding/heads to ensure parallelisation and finally preserve the input shape. The products of these matrices in each of the n heads, resulting in n Z matrices, are concatenated and multiplied by W_O to give the final output Z. The output Z contains the modified contextual embeddings of the 2 input words. (eg. row 1 would be the modified embedding of word 1)



This entire process for a sequence of words is represented like (credits: 3blue1brown):



Upper-triangular softmax(Q, K) serves as weights for V to be added onto embeddings.

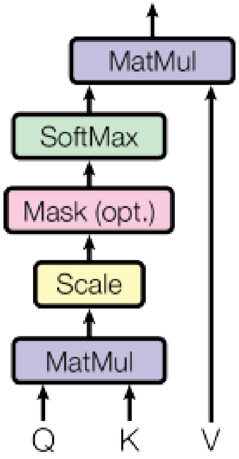
Next, these contextual embeddings are passed on to the ***DECODER*** which understands them, and generates output sequences.

Here's how the decoder works given encoder outputs Z - generating output tokens one at a time using the following components:

1. **Input Embedding + Positional Encoding**
   The decoder input (previously generated tokens) is embedded and combined with positional encodings.

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

2. **Masked Multi-Head Self-Attention**
   Each position in the output can only attend to earlier positions (causal masking). This prevents the model from "cheating" by looking ahead during training or inference.



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

$$\mathrm{MultiHead}(Q,K,V) = \mathrm{Concat}(\mathrm{head_1},...,\mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

3. **Cross-Attention (Encoder-Decoder Attention)**
   This layer lets each decoder token attend to all encoder outputs (Z). The decoder queries (Q) come from its previous layer, while keys (K) and values (V) come from the encoder's Z.

4. **Feed-Forward Network**
   A position-wise MLP with non-linearity, applied to each token.

5. **Layer Normalization & Residual Connections**
   Each sub-layer is wrapped with residual connections followed by layer normalization.

6. **Final Linear + Softmax Layer**
   The final output is passed through a linear layer projecting it to the vocabulary size, followed by softmax to get a probability distribution over the next word.

This encoder decoder approach, which dealt away with recurrence and convolutions and allowed parallelised temporal understanding, can be used for Sentiment Analysis, Named Entity Recognition (NER), Part-of-Speech Tagging, Question Answering, Language Modeling, Text

Generation, Summarization, Paraphrasing, Translation, Conversational AI, Text Classification and Information Retrieval. Here’s a parallel between Transformers and its preceding Seq2Seq models:

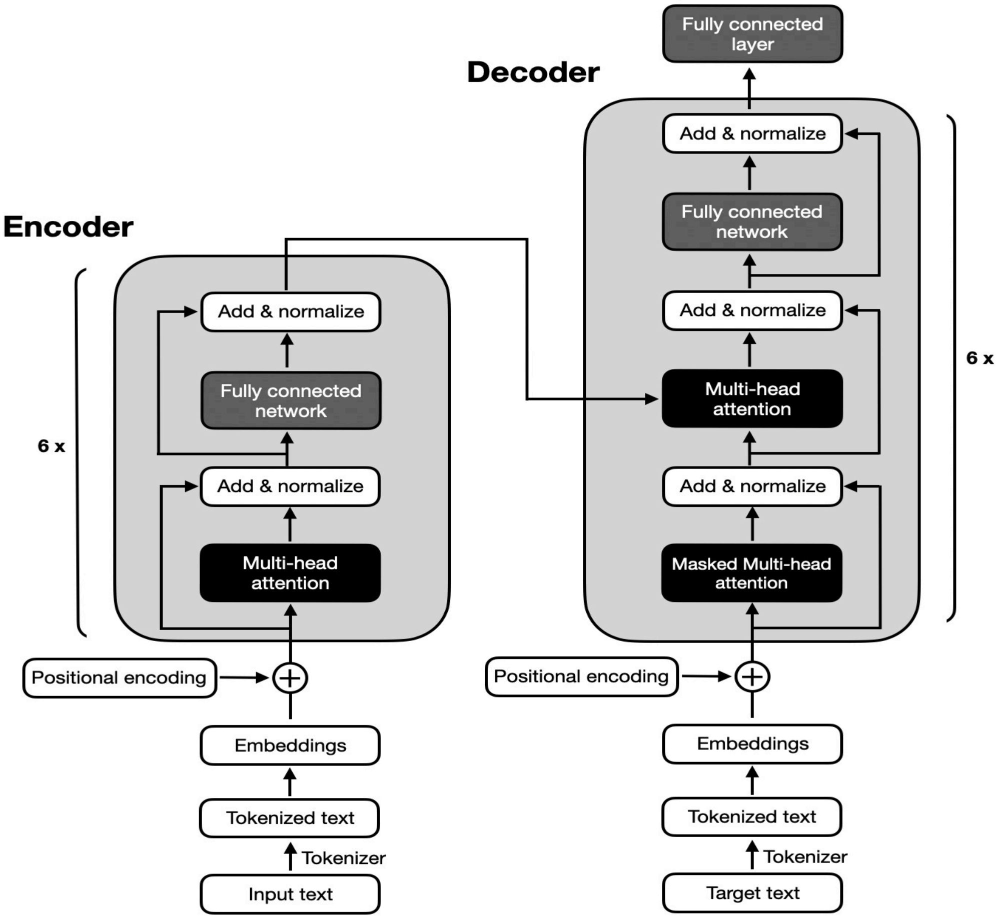
Component	Role	Function
Encoder	Reads and encodes the input sequence	Builds vector representations for each position
Decoder	Generates the output sequence, one element at a time	Uses encoder output and prior outputs (auto regression)
Auto-Regressive	Each output depends on previously generated ones	Ensures relevant sequences which are grammatically coherent
Transformer	Implements both with stacks of attention + feed-forward	Enables full parallelism and rich context

The core idea is self-attention across encoder and decoder, where each token attends to every other token in the sequence, directly capturing dependencies regardless of distance. This is unlike RNNs, which only implicitly pass information sequentially.

Key Innovations

- Multi-Head Attention: Instead of a single attention operation, the model splits query, key, and value vectors into multiple “heads” (e.g., 8). Each head learns to focus on different aspects of the sequence simultaneously. Their outputs are concatenated and linearly projected back, increasing expressive power.
- Positional Encoding: As the Transformer lacks recurrence, it adds positional encodings to input embeddings to provide information about token order, utilizing sine and cosine functions of different frequencies.

- Parallel Processing: Unlike RNNs, Transformers process all tokens simultaneously at each layer, enabling dramatically faster training and better hardware utilization.



1. Tokenization

Convert input text into subword tokens using a tokenizer (e.g., BPE or WordPiece).
"The cat" → ["The", "Ġcat"] → [101, 203] (token IDs)

2. Embedding + Positional Encoding

- Each token ID is converted to a dense vector using an embedding matrix.
- A learned positional encoding is added to give the model a sense of word order.

final_input = token_embedding + position_embedding

3. Transformer Block (repeated N times)

Each block contains:

a. Masked Multi-Head Self-Attention

- Calculates attention between all tokens seen so far.
- Masking prevents the model from accessing future tokens during training.
- Multiple "heads" allow focusing on different positions simultaneously.

b. Add & LayerNorm

- Add residual connection (skip connection) from input to output of attention.
- Apply layer normalization.

c. Feed-Forward Network (FFN)

- A two-layer dense network with a non-linearity (ReLU or GELU).
- Applies independently to each position (token).

d. Add & LayerNorm

- Add residual connection from input of FFN to its output.
- Apply layer normalization again.

4. Final Hidden States

- After all transformer blocks, each input token has a corresponding final hidden vector.
- For text generation, we typically use the last token's hidden state, and then progress auto-regressively. Factual information is stored in the weights of the intermediate FC layers. *(eg. The vector for Michael Jordan should fire the connections which lead to the embedding of Basketball/other personal info)*

5. Language Modeling Head

- The last hidden vector is projected into vocabulary space:

logits = hidden_state @ W + b

- A softmax is applied to produce a probability distribution over the vocabulary.
- The most likely next token is sampled or selected via argmax.

6. Autoregressive Generation

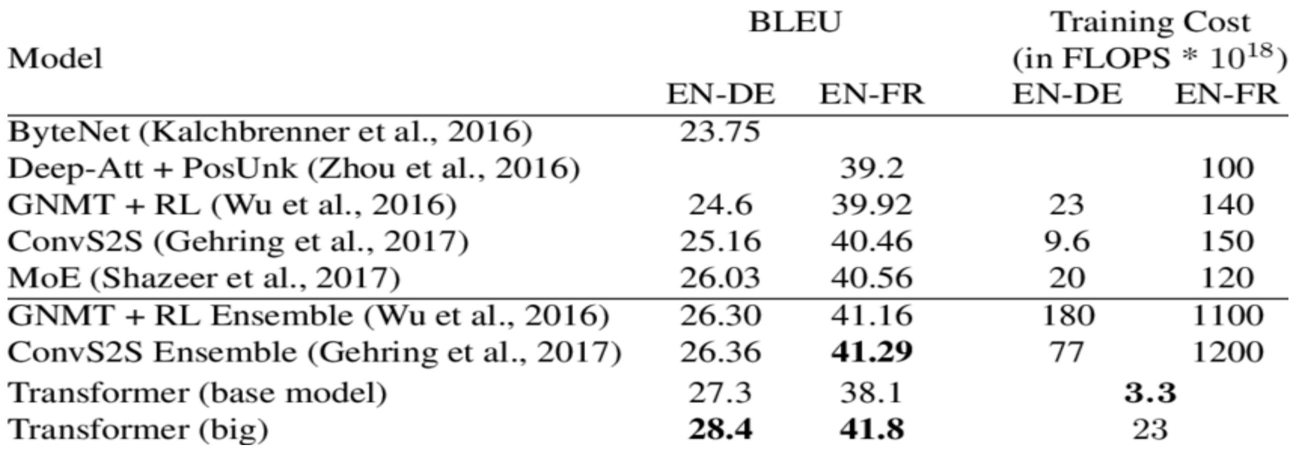
- The predicted token is appended to the input.
- The process is repeated to generate one token at a time.

7. Optional Decoding Strategies

Instead of always picking the highest probability token (greedy decoding), other methods can improve output diversity, especially with a parameter called temperature applied to the output softmax:

- Top-k sampling
- Top-p (nucleus) sampling
- Temperature scaling
- Beam search

Impact and Conclusion



Model	BLEU		Training Cost (in FLOPS * 10^{18})	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet (Kalchbrenner et al., 2016)	23.75			
Deep-Att + PosUnk (Zhou et al., 2016)		39.2		100
GNMT + RL (Wu et al., 2016)	24.6	39.92	23	140
ConvS2S (Gehring et al., 2017)	25.16	40.46	9.6	150
MoE (Shazeer et al., 2017)	26.03	40.56	20	120
GNMT + RL Ensemble (Wu et al., 2016)	26.30	41.16	180	1100
ConvS2S Ensemble (Gehring et al., 2017)	26.36	**41.29**	77	1200
Transformer (base model)	27.3	38.1	**3.3**	
Transformer (big)	**28.4**	**41.8**	23	

Transformers achieve better BLEU scores than previous sequential models

Transformers have reshaped the field of NLP, outperforming previous architectures such as RNNs, GRUs, and LSTMs across nearly every benchmark task. Their ability to model long-range dependencies, ace parallel computation, and scale to massive datasets has enabled breakthroughs in language understanding and generation. Models built on transformer architectures such as BERT, GPT, and T5 have set new SOTA results in tasks like question answering, machine translation, summarization, and text generation.

Unlike earlier models that struggled with context or required sequential processing, transformers used self-attention mechanisms to fix contextual importance to each token in a sequence, regardless of position. This led to more nuanced and semantically aware representations of language. Moreover, transformers' flexibility has extended their influence far beyond NLP, powering models in vision, speech, and even protein folding.

In summary, transformers not only surpassed previous model performance but also introduced a scalable, general-purpose framework that sparked the foundation of modern AI language and sequence modelling systems.