Donate
Create account
Log in

Transformer (deep learning architecture)

From Wikipedia, the free encyclopedia

Part of a series on
Machine learning and data mining

Paradigms
Supervised learning · Unsupervised learning · Semi-supervised learning · Self-supervised learning · Reinforcement learning · Meta-learning · Online learning · Batch learning · Curriculum learning · Rule-based learning · Neuro-symbolic AI · Neuromorphic engineering · Quantum machine learning

Problems
Classification · Generative modeling · Regression · Clustering · Dimensionality reduction · Density estimation · Anomaly detection · Data cleaning · AutoML · Association rules · Semantic analysis · Structured prediction · Feature engineering · Feature learning · Learning to rank · Grammar induction · Ontology learning · Multimodal learning

Supervised learning
(**classification** • **regression**)
Apprenticeship learning · Decision trees · Ensembles (Bagging · Boosting · Random forest) · *k*-NN · Linear regression · Naive Bayes · Artificial neural networks · Logistic regression · Perceptron · Relevance vector machine (RVM) · Support vector machine (SVM)

Clustering
BIRCH · CURE · Hierarchical · *k*-means · Fuzzy · Expectation–maximization (EM) · DBSCAN · OPTICS · Mean shift

Dimensionality reduction
Factor analysis · CCA · ICA · LDA · NMF · PCA · PGD · t-SNE · SDL

Structured prediction
Graphical models (Bayes net · Conditional random field · Hidden Markov)

Anomaly detection
RANSAC · *k*-NN · Local outlier factor · Isolation forest

Neural networks
Autoencoder · Deep learning · Feedforward neural network · Recurrent neural network (LSTM · GRU · ESN · reservoir computing) · Boltzmann machine (Restricted) · GAN · Diffusion model · SOM · Convolutional neural network (U-Net · LeNet · AlexNet · DeepDream) · Neural radiance field · **Transformer** (Vision) · Mamba · Spiking neural network · Memtransistor · Electrochemical RAM (ECRAM)

Reinforcement learning
Q-learning · SARSA · Temporal difference (TD) · Multi-agent (Self-play)

Learning with humans
Active learning · Crowdsourcing · Human-in-the-loop · Mechanistic interpretability · RLHF

Model diagnostics
Coefficient of determination · Confusion matrix · Learning curve · ROC curve

Mathematical foundations
Kernel machines · Bias–variance tradeoff · Computational learning theory · Empirical risk minimization · Occam learning · PAC learning · Statistical learning · VC theory · Topological deep learning

Journals and conferences
ECML PKDD · NeurIPS · ICML · ICLR · IJCAI · ML · JMLR

Related articles
Glossary of artificial intelligence · List of datasets for machine-learning research (List of datasets in computer vision and image processing) · Outline of machine learning

v · t · e

In deep learning, **transformer** is an architecture based on the multi-head attention mechanism, in which text is converted to numerical representations called tokens,

and each token is converted into a vector via lookup from a word embedding table.[1] At each layer, each token is then contextualized within the scope of the context window with other (unmasked) tokens via a parallel multi-head attention mechanism, allowing the signal for key tokens to be amplified and less important tokens to be diminished.

Transformers have the advantage of having no recurrent units, therefore requiring less training time than earlier recurrent neural architectures (RNNs) such as long short-term memory (LSTM).[2] Later variations have been widely adopted for training large language models (LLMs) on large (language) datasets.[3]

The modern version of the transformer was proposed in the 2017 paper "Attention Is All You Need" by researchers at Google.[1] Transformers were first developed as an improvement over previous architectures for machine translation,[4][5] but have found many applications since. They are used in large-scale natural language processing, computer vision (vision transformers), reinforcement learning,[6][7] audio,[8] multimodal learning, robotics,[9] and even playing chess.[10] It has also led to the development of pre-trained systems, such as generative pre-trained transformers (GPTs)[11] and BERT[12] (bidirectional encoder representations from transformers).

A standard Transformer architecture, showing on the left an encoder, and on the right a decoder. Note: it uses the pre-LN convention, which is different from the post-LN convention used in the original 2017 Transformer.

History [edit]

See also: Timeline of machine learning

Predecessors [edit]

For many years, sequence modelling and generation was done by using plain recurrent neural networks (RNNs). A well-cited early example was the Elman network (1990). In theory, the information from one token can propagate arbitrarily far down the sequence, but in practice the vanishing-gradient problem leaves the model's state at the end of a long sentence without precise, extractable information about preceding tokens.

A key breakthrough was LSTM (1995),[note 1] a RNN which used various innovations to overcome the vanishing gradient problem, allowing efficient learning of long-sequence modelling. One key innovation was the use of an attention mechanism which used neurons that multiply the outputs of other neurons, so-called *multiplicative units*.[13] Neural networks using multiplicative units were later called *sigma-pi networks*[14] or *higher-order networks*.[15] LSTM became the standard architecture for long sequence modelling until the 2017 publication of Transformers. However, LSTM still used sequential processing, like most other RNNs.[note 2] Specifically, RNNs operate one token at a time from first to last; they cannot operate in parallel over all tokens in a sequence.

Modern Transformers overcome this problem, but unlike RNNs, they require computation time that is quadratic in the size of the context window. The linearly scaling fast weight controller (1992) learns to compute a weight matrix for further processing depending on the input.[16] One of its two networks has "fast weights" or "dynamic links" (1981).[17][18][19] A slow neural network learns by gradient descent to generate keys and values for computing the weight changes of the fast neural network which computes answers to queries.[16] This was later shown to be equivalent to the unnormalized linear Transformer.[20][21]

Attention with seq2seq [edit]

Main article: Seq2seq § History

The idea of encoder-decoder sequence transduction had been developed in the early 2010s; commonly cited as the originators that produced seq2seq are two concurrently published papers from 2014.[22][23]

A 380M-parameter model for machine translation uses two long short-term memories (LSTM).[23] Its architecture consists of two parts. The *encoder* is an LSTM that takes in a sequence of tokens and turns it into a vector. The *decoder* is another LSTM that converts the vector into a sequence of tokens. Similarly, another 130M-parameter model used gated recurrent units (GRU) instead of LSTM.[22] Later research showed that GRUs are neither better nor worse than LSTMs for seq2seq.[24][25]

These early seq2seq models had no attention mechanism, and the state vector is accessible only after the *last* word of the source text was processed. Although in theory such a vector retains the information about the whole original sentence, in practice the information is poorly preserved. This is because the input is processed sequentially by one recurrent network into a *fixed*-size output vector, which is then processed by another recurrent network into an output. If the input is long, then the output vector would not be able to contain all relevant information, degrading the output. As evidence, reversing the input sentence improved seq2seq translation.[26]

The *RNNsearch* model introduced an attention mechanism to seq2seq for machine translation to solve the bottleneck problem (of the *fixed-size* output vector), allowing the model to process long-distance dependencies more easily. The name is because it "emulates searching through a source sentence during decoding a translation".[4]

The relative performances were compared between global (that of *RNNsearch*) and local (sliding window) attention model architectures for machine translation, finding that mixed attention had higher quality than global attention, while local attention reduced translation time.[27]

In 2016, Google Translate was revamped to Google Neural Machine Translation, which replaced the previous model based on statistical machine translation. The new model was a seq2seq model where the encoder and the decoder were both 8 layers of bidirectional LSTM.[28] It took nine months to develop, and it outperformed the statistical approach, which took ten years to develop.[29]

Parallelizing attention [edit]

Main article: Attention (machine learning) § History

Seq2seq models with attention (including self-attention) still suffered from the same issue with recurrent networks, which is that they are hard to parallelize, which prevented them from being accelerated on GPUs. In 2016, *decomposable attention* applied a self-attention mechanism to feedforward networks, which are easy to parallelize, and achieved SOTA result in textual entailment with an order of magnitude fewer parameters than LSTMs.[30] One of its authors, Jakob Uszkoreit, suspected that attention *without* recurrence would be sufficient for language translation, thus the title "attention is *all* you need".[31] That hypothesis was against conventional wisdom at the time, and even his father Hans Uszkoreit, a well-known computational linguist, was skeptical.[31] In the same year, self-attention (called *intra-attention or intra-sentence attention*) was proposed for LSTMs.[32]

In 2017, the original (100M-sized) encoder-decoder transformer model was proposed in the "Attention is all you need" paper. At the time, the focus of the research was on improving seq2seq for machine translation, by removing its recurrence to process all tokens in parallel, but preserving its dot-product attention mechanism to keep its text processing performance.[1] This led to the introduction of a multi-head attention model that was easier to parallelize due to the use of independent heads and the lack of recurrence. Its parallelizability was an important factor to its widespread use in large neural networks.[33]

AI boom era [edit]

Already in spring 2017, even before the "Attention is all you need" preprint was published, one of the co-authors applied the "decoder-only" variation of the architecture to generate fictitious Wikipedia articles.[34] Transformer architecture is now used alongside many generative models that contribute to the ongoing AI boom.

In language modelling, ELMo (2018) was a bi-directional LSTM that produces contextualized word embeddings, improving upon the line of research from bag of words and word2vec. It was followed by BERT (2018), an encoder-only Transformer model.[35] In 2019 October, Google started using BERT to process search queries.[36] In 2020, Google Translate replaced the previous RNN-encoder–RNN-decoder model by a Transformer-encoder–RNN-decoder model.[37]

Starting in 2018, the OpenAI GPT series of decoder-only Transformers became state of the art in natural language generation. In 2022, a chatbot based on GPT-3, ChatGPT, became unexpectedly[38] popular, triggering a boom around large language models.[39][40]

Since 2020, Transformers have been applied in modalities beyond text, including the vision transformer,[41] speech recognition,[42] robotics,[6] and multimodal.[43] The vision transformer, in turn, stimulated new developments in convolutional neural networks.[44] Image and video generators like DALL-E (2021), Stable Diffusion 3 (2024),[45] and Sora (2024), use Transformers to analyse input data (like text prompts) by breaking it down into "tokens" and then calculating the relevance between each token using self-attention, which helps the model understand the context and relationships within the data.

Training [edit]

Methods for stabilizing training [edit]

The plain transformer architecture had difficulty converging. In the original paper[1] the authors recommended using learning rate warmup. That is, the learning rate should linearly scale up from 0 to maximal value for the first part of the training (usually recommended to be 2% of the total number of training steps), before decaying again.

A 2020 paper found that using layer normalization *before* (instead of after) multiheaded attention and feedforward layers stabilizes training, not requiring learning rate warmup.[46]

Pretrain-finetune [edit]

Transformers typically are first pretrained by self-supervised learning on a large generic dataset, followed by supervised fine-tuning on a small task-specific dataset. The pretrain dataset is typically an unlabeled large corpus, such as The Pile. Tasks for pretraining and fine-tuning commonly include:

- language modeling[12]
- next-sentence prediction[12]
- question answering[3]
- reading comprehension
- sentiment analysis[1]
- paraphrasing[1]

The T5 transformer report[47] documents a large number of natural language pretraining tasks. Some examples are:

- restoring or repairing incomplete or corrupted text. For example, the input, *"Thank you ~~ me to your party ~~ week",* might generate the output, *"Thank you **for inviting** me to your party **last** week".*
- translation between natural languages (machine translation)
- judging the pragmatic acceptability of natural language. For example, the following sentence might be judged "not acceptable",[48] because even though it is syntactically well-formed, it is improbable in ordinary human usage: *The course is jumping well.*

Note that while each of these tasks is trivial or obvious for human native speakers of the language (or languages), they have typically proved challenging for previous generations of machine learning architecture.

Tasks [edit]

See also: Large language model § Evaluation

In general, there are 3 classes of language modelling tasks: "masked",[49] "autoregressive",[50] and "prefixLM".[51] These classes are independent of a specific modeling architecture such as Transformer, but they are often discussed in the context of Transformer.

In a masked task,[49] one or more of the tokens is masked out, and the model would produce a probability distribution predicting what the masked-out tokens are based on the context. The loss function for the task is typically sum of log-perplexities for the masked-out tokens:

$$\text{Loss} = -\sum_{t \in \text{masked tokens}} \ln(\text{probability of } t \text{ conditional on its context})$$

and the model is trained to minimize this loss function. The BERT series of models are trained for masked token prediction and another task.

In an autoregressive task,[50] the entire sequence is masked at first, and the model produces a probability distribution for the first token. Then the first token is revealed and the model predicts the second token, and so on. The loss function for the task is still typically the same. The GPT series of models are trained by autoregressive tasks.

In a prefixLM task,[51] the sequence is divided into two parts. The first part is presented as context, and the model predicts the first token of the second part. Then that would be revealed, and the model predicts the second token, and so on. The loss function for the task is still typically the same. The T5 series of models are trained by prefixLM tasks.

Note that "masked" as in "masked language modelling" is not "masked" as in "masked attention", and "prefixLM" (prefix language modeling) is not "prefixLM" (prefix language model).

Architecture [edit]

All transformers have the same primary components:

- Tokenizers, which convert text into tokens.
- Embedding layer, which converts tokens and positions of the tokens into vector representations.
- Transformer layers, which carry out repeated transformations on the vector representations, extracting more and more linguistic information. These consist of alternating attention and feedforward layers. There are two major types of transformer layers: encoder layers and decoder layers, with further variants.
- Un-embedding layer, which converts the final vector representations back to a probability distribution over the tokens.

The following description follows exactly the Transformer as described in the original paper. There are variants, described in the following section.

By convention, we write all vectors as row vectors. This, for example, means that pushing a vector through a linear layer means multiplying it by a weight matrix on the right, as xW.

Tokenization [edit]

Main article: Lexical analysis

As the Transformer architecture natively processes numerical data, not text, there must be a translation between text and tokens. A token is an integer that represents a character, or a short segment of characters. On the input side, the input text is parsed into a token sequence. Similarly, on the output side, the output tokens are parsed back to text. The module doing the conversion between texts and token sequences is a tokenizer.

The set of all tokens is the vocabulary of the tokenizer, and its size is the *vocabulary size* $n_{\text{vocabulary}}$. When faced with tokens outside the vocabulary, typically a special token is used, written as "[UNK]" for "unknown".

Some commonly used tokenizers are byte pair encoding, WordPiece, and SentencePiece.

Embedding [edit]

Further information: Word embedding

Each token is converted into an embedding vector via a lookup table. Equivalently stated, it multiplies a one-hot representation of the token by an embedding matrix M. For example, if the input token is 3, then the one-hot representation is $[0,0,0,1,0,0,\dots]$, and its embedding vector is

$$\text{Embed}(3) = [0,0,0,1,0,0,\dots]M$$

The token embedding vectors are added to their respective positional encoding vectors (see below), producing the sequence of input vectors.

The number of dimensions in an embedding vector is called *hidden size* or *embedding size* and written as d_{emb}.[35] This size is written as d_{model} in the original Transformer paper.[1]

Un-embedding [edit]

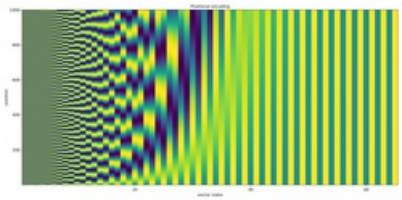
An un-embedding layer is almost the reverse of an embedding layer. Whereas an embedding layer converts a token into a vector, an un-embedding layer converts a vector into a probability distribution over tokens.

The un-embedding layer is a linear-softmax layer:

The matrix has shape $(d_{\text{emb}}, n_{\text{vocabulary}})$. The embedding matrix M and the un-embedding matrix W are sometimes required to be transposes of each other, a practice called weight tying.[52]

Positional encoding [edit]

A positional encoding is a fixed-size vector representation of the relative positions of tokens within a sequence: it provides the transformer model with information about *where* the words are in the input sequence. This induces a bias towards the order of the input sequence, so that, for example, the input sequence "man bites dog" is processed differently from "dog bites man".



A diagram of a sinusoidal positional encoding with parameters $N = 10000, d = 100$

The positional encoding is defined as a function of type $f : \mathbb{R} \to \mathbb{R}^d; d \in \mathbb{Z}, d > 0$, where d is a positive even integer. The full positional encoding defined in the original paper[1] is:

$$(f(t)_{2k}, f(t)_{2k+1}) = (\sin(\theta), \cos(\theta)) \quad \forall k \in \{0, 1, \dots, d/2 - 1\}$$

where $\theta = \frac{t}{r^k}, r = N^{2/d}$.

Here, N is a free parameter that should be significantly larger than the biggest k that would be input into the positional encoding function. The original paper uses $N = 10000$.

The function is in a simpler form when written as a complex function of type $f : \mathbb{R} \to \mathbb{C}^{d/2}$

$$f(t) = \left(e^{it/r^k}\right)_{k=0,1,\dots,\frac{d}{2}-1}$$

where $r = N^{2/d}$.

The main reason for using this positional encoding function is that using it, shifts are linear transformations:

$$f(t + \Delta t) = \text{diag}(f(\Delta t))f(t)$$

where $\Delta t \in \mathbb{R}$ is the distance one wishes to shift. This allows the transformer to take any encoded position, and find the encoding of the position n-steps-ahead or n-steps-behind, by a matrix multiplication.

By taking a linear sum, any convolution can also be implemented as linear transformations:

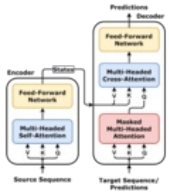
$$\sum_j c_j f(t + \Delta t_j) = \left(\sum_j c_j \,\text{diag}(f(\Delta t_j))\right) f(t)$$

for any constants c_j. This allows the transformer to take any encoded position and find a linear sum of the encoded locations of its neighbors. This sum of encoded positions, when fed into the attention mechanism, would create attention weights on its neighbors, much like what happens in a convolutional neural network language model. In the author's words, "we hypothesized it would allow the model to easily learn to attend by relative position."

In typical implementations, all operations are done over the real numbers, not the complex numbers, but since complex multiplication can be implemented as real 2-by-2 matrix multiplication, this is a mere notational difference.

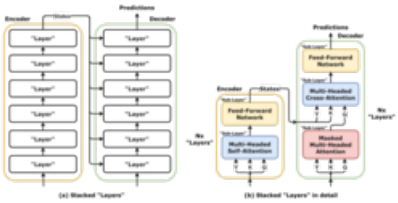
Encoder-decoder (overview) [edit]

Like earlier seq2seq models, the original transformer model used an **encoder-decoder** architecture. The encoder consists of encoding layers that process all the input tokens together one layer after another, while the decoder consists of decoding layers that iteratively process the encoder's output and the decoder's output tokens so far.



One encoder-decoder block

The purpose of each encoder layer is to create contextualized representations of the tokens, where each representation corresponds to a token that "mixes" information from other input tokens via self-attention mechanism. Each decoder layer contains two attention sublayers: (1) cross-attention for incorporating the output of encoder (contextualized input token representations), and (2) self-attention for "mixing" information among the input tokens to the decoder (i.e. the tokens generated so far during inference time).[53][54]



A Transformer is composed of stacked encoder layers and decoder layers.

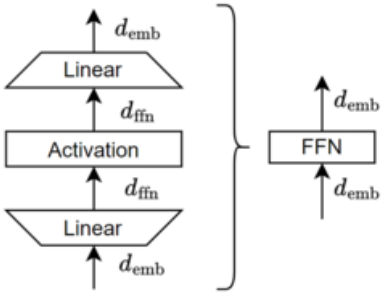
Both the encoder and decoder layers have a feed-forward neural network for additional processing of their outputs and contain residual connections and layer normalization steps.[54] These feed-forward layers contain most of the parameters in a Transformer model.

Feedforward network [edit]

The feedforward network (FFN) modules in a Transformer are 2-layered multilayer perceptrons:

$$\mathrm{FFN}(x) = \phi(xW^{(1)} + b^{(1)})W^{(2)} + b^{(2)}$$

where $W^{(1)}$ and $W^{(2)}$ are weight matrices and $b^{(1)}$ and $b^{(2)}$ are bias vectors, and ϕ is its activation function. The original Transformer used ReLU activation.



The feedforward network module. It is a two-layered network that maps d_{emb}-dimensional vectors into d_{emb}-dimensional vectors.

The number of neurons in the middle layer is called *intermediate size* (GPT),[55] *filter size* (BERT),[35] or *feedforward size* (BERT).[35] It is typically larger than the embedding size. For example, in both GPT-2 series and BERT series, the intermediate size of a model is 4 times its embedding size: $d_{\text{ffn}} = 4d_{\text{emb}}$.

Scaled dot-product attention [edit]

Main article: Dot-product attention

Attention head [edit]

The attention mechanism used in the Transformer architecture are scaled dot-product attention units. For each unit, the transformer model learns three weight matrices: the query weights W^Q, the key weights W^K, and the value weights W^V.

The module takes three sequences, a query sequence, a key sequence, and a value sequence. The query sequence is a sequence of length $l_{\text{seq, query}}$, and each entry is a vector of dimension $d_{\text{emb, query}}$. Similarly for the key and value sequences.

For each vector $x_{i,\text{query}}$ in the query sequence, it is multiplied by a matrix W^Q to produce a query vector $q_i = x_{i,\text{query}}W^Q$. The matrix of all query vectors is the query matrix:

$$Q = X_{\text{query}}W^Q$$

Similarly, we construct the key matrix $K = X_{\text{key}}W^K$ and the value matrix $V = X_{\text{value}}W^V$.

It is usually the case that all W^Q, W^K, W^V are square matrices, meaning $d_{\text{emb, query}} = d_{\text{query}}$, etc.

Attention weights are calculated using the query and key vectors: the attention weight a_{ij} from token i to token j is the dot product between q_i and k_j. The attention weights are divided by the square root of the dimension of the key vectors, $\sqrt{d_k}$, which stabilizes gradients during training, and passed through a softmax which

normalizes the weights. The fact that W^Q and W^K are different matrices allows attention to be non-symmetric: if token i attends to token j (i.e. $q_i \cdot k_j$ is large), this does not necessarily mean that token j will attend to token i (i.e. $q_j \cdot k_i$ could be small). The output of the attention unit for token i is the weighted sum of the value vectors of all tokens, weighted by a_{ij}, the attention from token i to each token.

The attention calculation for all tokens can be expressed as one large matrix calculation using the softmax function, which is useful for training due to computational matrix operation optimizations that quickly compute matrix operations. The matrices Q, K and V are defined as the matrices where the ith rows are vectors q_i, k_i, and v_i respectively. Then we can represent the attention as

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

where the softmax is applied over each of the rows of the matrix.

The number of dimensions in a query vector is *query size* d_{query} and similarly for the *key size* d_{key} and *value size* d_{value}. The output dimension of an attention head is its *head dimension* d_{head}. The attention mechanism requires the following three equalities to hold:

$$l_{\text{seq, key}} = l_{\text{seq, value}},\ d_{\text{query}} = d_{\text{key}},\ d_{\text{value}} = d_{\text{head}}$$

but is otherwise unconstrained.

If the attention head is used in a self-attention fashion, then $X_{\text{query}} = X_{\text{key}} = X_{\text{value}}$. If the attention head is used in a cross-attention fashion, then usually $X_{\text{query}} \neq X_{\text{key}} = X_{\text{value}}$. It is theoretically possible for all three to be different, but that is rarely the case in practice.

Multiheaded attention [edit]

One set of (W^Q, W^K, W^V) matrices is called an *attention head*, and each layer in a transformer model has multiple attention heads. While each attention head attends to the tokens that are relevant to each token, multiple attention heads allow the model to do this for different definitions of "relevance". Specifically, the query and key projection matrices, W^Q and W^K, which are involved in the attention score computation, defines the "relevance". Meanwhile, the value projection matrix W^V, in combination with the part of the output projection matrix W^O, determines how the attended tokens influence what information is passed to subsequent layers and ultimately the output logits. In addition, the scope of attention, or the range of token relationships captured by each attention head, can expand as tokens pass through successive layers. This allows the model to capture more complex and long-range dependencies in deeper layers. Many transformer attention heads encode relevance relations that are meaningful to humans. For example, some attention heads can attend mostly to the next word, while others mainly attend from verbs to their direct objects.[56] The computations for each attention head can be performed in parallel, which allows for fast processing. The outputs for the attention layer are concatenated to pass into the feed-forward neural network layers.

Concretely, let the multiple attention heads be indexed by i, then we have

$$\text{MultiheadedAttention}(Q, K, V) = \text{Concat}_{i \in [n_{\text{heads}}]}(\text{Attention}(XW_i^Q, XW_i^K, XW_i^V))W^O$$

where the matrix X is the concatenation of word embeddings, and the matrices W_i^Q, W_i^K, W_i^V are "projection matrices" owned by individual attention head i, and W^O is a final projection matrix owned by the whole multi-headed attention head.

It is theoretically possible for each attention head to have a different head dimension d_{head}, but that is rarely the case in practice.

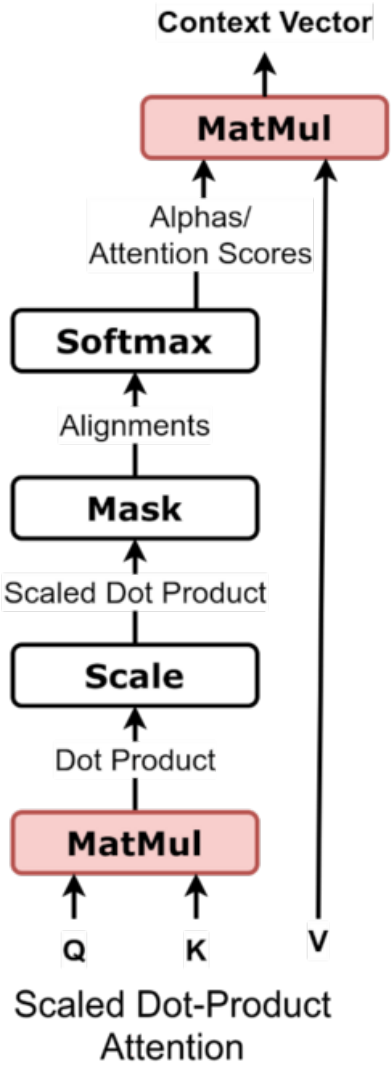
As an example, in the smallest GPT-2 model, there are only self-attention mechanisms. It has the following dimensions:

$$d_{\text{emb}} = 768, n_{\text{head}} = 12, d_{\text{head}} = 64$$

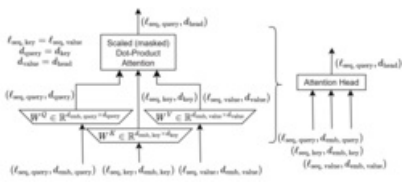
Since $12 \times 64 = 768$, its output projection matrix $W^O \in \mathbb{R}^{(12\times 64)\times 768}$ is a square matrix.

Masked attention [edit]

The Transformer architecture is constructed to calculate output tokens iteratively. Assuming $t = 0$ refers to the calculation of the first output token $i = 0$, for step $t > 0$, the output token $i = 0$ shall remain constant. This ensures properties of the model similar



Scaled dot-product attention, block diagram



Exact dimension counts within an attention head module

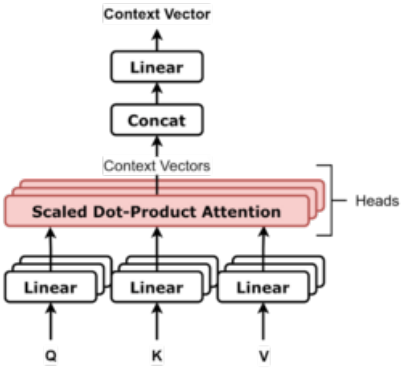
to autoregressive models.[1] Therefore, at every time step t, the calculation for all outputs y_i should not have access to tokens at position j for $j>=i$ (as it naturally is the case for time step $t=i$, when tokens $j>t$ are not yet calculated). This behavior may be accomplished before the softmax stage by adding a mask matrix M that is $-\infty$ at entries where the attention link must be cut, and 0 at other places:

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

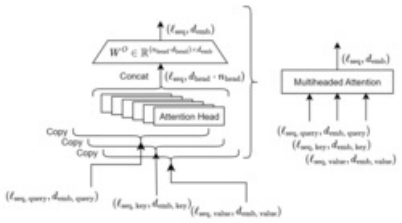
The following matrix is commonly used in decoder self-attention modules, called "causal masking":

$$M_{\text{causal}} = \begin{bmatrix} 0 & -\infty & -\infty & \dots & -\infty \\ 0 & 0 & -\infty & \dots & -\infty \\ 0 & 0 & 0 & \dots & -\infty \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix}$$

In words, it means that each token can pay attention to itself, and every token before it, but not any after it. A non-masked attention module can be thought of as a masked attention module where the mask has all entries zero. As an example of an uncommon use of mask matrix, the XLNet considers all masks of the form $PM_{\text{causal}}P^{-1}$, where P is a random permutation matrix.[57]



Multiheaded attention, block diagram



Exact dimension counts within a multiheaded attention module

Encoder [edit]

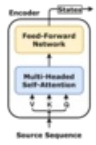
An encoder consists of an embedding layer, followed by multiple encoder layers.

Each encoder layer consists of two major components: a self-attention mechanism and a feed-forward layer. It takes an input as a sequence of input vectors, applies the self-attention mechanism, to produce an intermediate sequence of vectors, then applies the feed-forward layer for each vector individually. Schematically, we have:

given input vectors $h_0, h_1, \dots$

combine them into a matrix $H = \begin{bmatrix} h_0 \\ h_1 \\ \vdots \end{bmatrix}$

$$\text{EncoderLayer}(H) = \begin{bmatrix} \text{FFN}(\text{MultiheadedAttention}(H,H,H)_0) \\ \text{FFN}(\text{MultiheadedAttention}(H,H,H)_1) \\ \vdots \end{bmatrix}$$



One encoder layer

where FFN stands for "feed-forward network". We can more succinctly write it as

$$\text{EncoderLayer}(H) = \text{FFN}(\text{MultiheadedAttention}(H,H,H))$$

with the implicit convention that the FFN is applied to each row of the matrix individually.

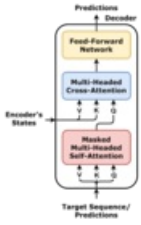
The encoder layers are stacked. The first encoder layer takes the sequence of input vectors from the embedding layer, producing a sequence of vectors. This sequence of vectors is processed by the second encoder, and so on. The output from the final encoder layer is then used by the decoder.

As the encoder processes the entire input all at once, every token can attend to every other token (all-to-all attention), so there is no need for causal masking.

Decoder [edit]

A decoder consists of an embedding layer, followed by multiple decoder layers, followed by an un-embedding layer.

Each decoder consists of three major components: a causally masked self-attention mechanism, a cross-attention mechanism, and a feed-forward neural network. The decoder functions in a similar fashion to the encoder, but an additional attention mechanism is inserted which instead draws relevant information from the encodings generated by the encoders. This mechanism can also be called the *encoder-decoder attention*.[1][54]



One decoder layer

Like the first encoder, the first decoder takes positional information and embeddings of the output sequence as its input, rather than encodings. The transformer must not use the current or future output to predict an output, so the output sequence must be partially masked to prevent this reverse information flow.[1] This allows for autoregressive text generation. For decoding, all-to-all attention is inappropriate, because a token cannot attend to tokens not yet generated. Thus, the self-attention module in the decoder is causally masked.

In contrast, the cross-attention mechanism attends to the output vectors of the encoder, which is computed before the decoder starts decoding. Consequently, there is no need for masking in the cross-attention mechanism.

Schematically, we have:

$$H' = \text{MaskedMultiheadedAttention}(H, H, H)$$
$$\text{DecoderLayer}(H) = \text{FFN}(\text{MultiheadedAttention}(H', H^E, H^E))$$

where H^E is the matrix with rows being the output vectors from the encoder.

The last decoder is followed by a final un-embedding layer. to produce the output probabilities over the vocabulary. Then, one of the tokens is sampled according to the probability, and the decoder can be run again to produce the next token, etc, autoregressively generating output text.

Adapted architectures [edit]

Many large language models, since they do not need to predict a whole new sequence from an input sequence, only use the encoder or decoder of the original transformer architecture. Early GPT models are decoder-only models trained to predict the next token in a sequence.[58] BERT, another language model, only makes use of an encoder, and is trained to predict a randomly masked token in a sequence.[35]

Full transformer architecture [edit]

Sublayers [edit]

Each encoder layer contains 2 sublayers: the self-attention and the feedforward network. Each decoder layer contains 3 sublayers: the causally masked self-attention, the cross-attention, and the feedforward network.

The final points of detail are the residual connections and layer normalization (LayerNorm, or LN), which while conceptually unnecessary, are necessary for numerical stability and convergence.

The residual connection, which is introduced to avoid vanishing gradient issues and stabilize the training process, can be expressed as follows: y = F(x) + x. The expression indicates that an output y is the sum of the transformation of input x (F(x)) and the input itself (x). Adding the input x can preserve the input information and avoid issues when the gradient of F(x) is close to zero.

Similarly to how the feedforward network modules are applied individually to each vector, the LayerNorm is also applied individually to each vector.

There are two common conventions in use: the *post-LN* and the *pre-LN* convention. In the post-LN convention, the output of each sublayer is

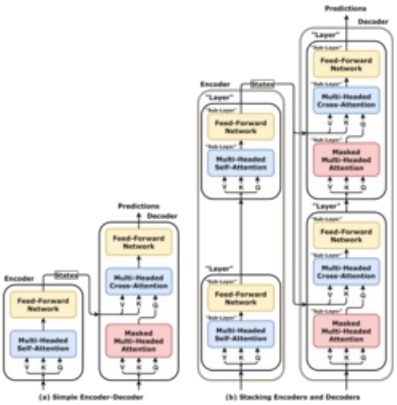
$$\text{LayerNorm}(x + \text{Sublayer}(x))$$

where Sublayer is the function implemented by the sublayer itself.

In the pre-LN convention, the output of each sublayer is

$$x + \text{Sublayer}(\text{LayerNorm}(x))$$

The original 2017 Transformer used the post-LN convention. It was difficult to train and required careful hyperparameter tuning and a "warm-up" in learning rate, where it starts small and gradually increases. The pre-LN convention, proposed several times in 2018,[59] was found to be easier to train, requiring no warm-up, leading to faster convergence.[46]



(a) One encoder layer and one decoder layer. (b) Two encoder layers and two decoder layers. The sublayers are labelled as well.

Transformer encoder with norm-first and norm-last

Pseudocode [edit]

The following is the pseudocode for a standard pre-LN encoder-decoder Transformer, adapted from[60]

```
input: Encoder input t_e
       Decoder input t_d
output: Array of probability distributions, with
shape (decoder vocabulary size x length(decoder
output sequence))

/* encoder */
z_e ← encoder.tokenizer(t_e)

for each t in 1:length(z_e) do
    z_e[t] ← encoder.embedding(z_e[t]) +
```

```
encoder.positional_embedding(t)

for each l in 1:length(encoder.layers) do
    layer ← encoder.layers[l]

    /* first sublayer */
    z_e_copy ← copy(z_e)
    for each t in 1:length(z_e) do
        z_e[t] ← layer.layer_norm(z_e[t])
    z_e ← layer.multiheaded_attention(z_e, z_e, z_e)
    for each t in 1:length(z_e) do
        z_e[t] ← z_e[t] + z_e_copy[t]

    /* second sublayer */
    z_e_copy ← copy(z_e)
    for each t in 1:length(z_e) do
        z_e[t] ← layer.layer_norm(z_e[t])
    z_e ← layer.feedforward(z_e)
    for each t in 1:length(z_e) do
        z_e[t] ← z_e[t] + z_e_copy[t]

for each t in 1:length(z_e) do
    z_e[t] ← encoder.final_layer_norm(z_e[t])

/* decoder */
z_d ← decoder.tokenizer(t_d)

for each t in 1:length(z_d) do
    z_d[t] ← decoder.embedding(z_d[t]) +
decoder.positional_embedding(t)

for each l in 1:length(decoder.layers) do
        layer ← decoder.layers[l]

        /* first sublayer */
        z_d_copy ← copy(z_d)
        for each t in 1:length(z_d) do
            z_d[t] ← layer.layer_norm(z_d[t])
        z_d ← layer.masked_multiheaded_attention(z_d,
z_d, z_d)
        for each t in 1:length(z_d) do
            z_d[t] ← z_d[t] + z_d_copy[t]

        /* second sublayer */
        z_d_copy ← copy(z_d)
        for each t in 1:length(z_d) do
            z_d[t] ← layer.layer_norm(z_d[t])
        z_d ← layer.multiheaded_attention(z_d, z_e,
z_e)
        for each i in 1:length(z_d) do
            z_d[t] ← z_d[t] + z_d_copy[t]

        /* third sublayer */
        z_d_copy ← copy(z_d)
        for each t in 1:length(z_d) do
            z_d[t] ← layer.layer_norm(z_d[t])
        z_d ← layer.feedforward(z_d)
        for each t in 1:length(z_d) do
            z_d[t] ← z_d[t] + z_d_copy[t]

z_d ← decoder.final_layer_norm(z_d)

output_distributions ← []
for each t in 1:length(z_d) do
```

Transformer decoder with norm-first and norm-last

Block diagram for the full Transformer architecture

Schematic object hierarchy for the full Transformer architecture, in object-oriented programming style

```
    output_distributions.append(decoder.unembed(z_d[t]))

return output_distributions
```

Terminology [edit]

The Transformer architecture, being modular, allows variations. Several common variations are described here.[61]

An "encoder-only" Transformer applies the encoder to map an input text into a sequence of vectors that represent the input text. This is usually used for text embedding and representation learning for downstream applications. BERT is encoder-only. They are less often used currently, as they were found to be not significantly better than training an encoder-decoder Transformer, then taking just the encoder.[51]

A "decoder-only" Transformer is not literally decoder-only, since without an encoder, the cross-attention mechanism has nothing to attend to. Thus, the decoder layers in a decoder-only Transformer is composed of just two sublayers: the causally masked self-attention, and the feedforward network. This is usually used for text generation and instruction following. The models in the GPT series and Chinchilla series are decoder-only.

An "encoder-decoder" Transformer is generally the same as the original Transformer, with 2 sublayers per encoder layer and 3 sublayers per decoder layer, etc. They might have minor architectural improvements, such as alternative activation functions, changing the location of normalization, etc. This is also usually used for text generation and instruction following. The models in the T5 series are encoder-decoder.[61]

A "prefixLM" (prefix language model) is a decoder-only architecture, but with prefix masking, which is different from causal masking. Specifically, it has mask of the form[61]: Figure 3

$$M_{\text{prefixLM}} = \begin{bmatrix} \mathbf{0} & -\infty \\ \mathbf{0} & M_{\text{causal}} \end{bmatrix}$$

where the first columns correspond to the "prefix", and the subsequent columns correspond to the autoregressively generated text based on the prefix. They resemble encoder-decoder models, but has less "sparsity". Such models are rarely used, though they are cited as theoretical possibilities and benchmarked comparisons.[51]

There are also mixed seq2seq models. For example, in 2020, Google Translate replaced the previous RNN-encoder–RNN-decoder model by a Transformer-encoder–RNN-decoder model, on the argument that an RNN-decoder runs much faster than Transformer-decoder when run autoregressively.[62]

Subsequent work [edit]

Alternative activation functions [edit]

The original transformer uses ReLU activation function. Other activation functions were developed. The Llama series and PaLM used SwiGLU;[63] both GPT-1 and BERT[35] used GELU.[64]

Alternative activation functions are often used in combination with Gated Linear Units in the feedforward module.[63]

Alternative normalizations [edit]

The normalization used in the Transformer can be different from LayerNorm. One example is RMSNorm[65] which is used in the Llama series. Other examples include CapsuleNorm[66] ScaleNorm,[67] or FixNorm.[67]

Alternative positional encodings [edit]

Transformers may use other positional encoding methods than sinusoidal.[68]

The original Transformer paper reported using a learned positional encoding,[69] but finding it not superior to the sinusoidal one.[1] Later,[70] found that causal masking itself provides enough signal to a Transformer decoder that it can learn to implicitly perform absolute positional encoding without the positional encoding module.

RoPE [edit]

RoPE (rotary positional embedding),[71] is best explained by considering a list of 2-dimensional vectors $[(x_1^{(1)}, x_1^{(2)}), (x_2^{(1)}, x_2^{(2)}), (x_3^{(1)}, x_3^{(2)}), \dots]$. Now pick some angle θ. Then RoPE encoding is

$$\text{RoPE}(x_m^{(1)}, x_m^{(2)}, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} x_m^{(1)} \\ x_m^{(2)} \end{pmatrix} = \begin{pmatrix} x_m^{(1)} \cos m\theta - x_m^{(2)} \sin m\theta \\ x_m^{(2)} \cos m\theta + x_m^{(1)} \sin m\theta \end{pmatrix}$$

Equivalently, if we write the 2-dimensional vectors as complex numbers $z_m := x_m^{(1)} + i x_m^{(2)}$, then RoPE encoding is just multiplication by an angle:

$$\mathrm{RoPE}(z_m, m) = e^{im\theta} z_m$$

For a list of $2n$-dimensional vectors, a RoPE encoder is defined by a sequence of angles $\theta^{(1)}, \dots, \theta^{(n)}$. Then the RoPE encoding is applied to each pair of coordinates.

The benefit of RoPE is that the dot-product between two vectors depends on their relative location only:

$$\mathrm{RoPE}(x, m)^T \mathrm{RoPE}(y, n) = \mathrm{RoPE}(x, m+k)^T \mathrm{RoPE}(y, n+k)$$

for any integer k.

ALiBi [edit]

ALiBi (Attention with Linear Biases)[72] is not a *replacement* for the positional encoder on the original transformer. Instead, it is an *additional* positional encoder that is directly plugged into the attention mechanism. Specifically, the ALiBi attention mechanism is

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

Here, s is a real number ("scalar"), and B is the *linear bias* matrix defined by

$$B = \begin{pmatrix} 0 & 1 & 2 & 3 & \cdots \\ -1 & 0 & 1 & 2 & \cdots \\ -2 & -1 & 0 & 1 & \cdots \\ -3 & -2 & -1 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

in other words, $B_{ij} = j - i$. The idea being that the linear bias matrix is a softened mask. Just as 0 represent full attention paid, and $-\infty$ represents no attention paid, the linear bias matrix increases attention paid in one direction and decreases attention paid in the other direction.

ALiBi allows pretraining on short context windows, then fine-tuning on longer context windows. Since it is directly plugged into the attention mechanism, it can be combined with any positional encoder that is plugged into the "bottom" of the entire network (which is where the sinusoidal encoder on the original transformer, as well as RoPE and many others, are located).

Relative Position Encodings [edit]

Relative Position Encodings[73] is similar to ALiBi, but more generic:

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

where B is a Toeplitz matrix, that is, $B_{i,j} = B_{i',j'}$ whenever $i - j = i' - j'$. This is contrasted with the original sinusoidal positional encoding, which is an "absolute positional encoding".[74]

Efficient implementation [edit]

The transformer model has been implemented in standard deep learning frameworks such as TensorFlow and PyTorch. *Transformers* is a library produced by Hugging Face that supplies transformer-based architectures and pretrained models.[11]

KV caching [edit]

When an autoregressive transformer is used for inference, such as generating text, the query vector is different at each step, but the already-computed key and value vectors are always the same. The **KV caching** method saves the computed key and value vectors at each attention block, so that they are not recomputed at each new token. **PagedAttention** applies memory paging to KV caching.[75][76][77]

If a transformer is used with a baked-in prompt, such as ["You are a customer support agent..."], then the key and value vectors can be computed for the prompt, and saved on disk. The saving in compute is significant when the model is used for many short interactions, such as in online chatbots.

FlashAttention [edit]

FlashAttention[78] is an algorithm that implements the transformer attention mechanism efficiently on a GPU. It is a communication-avoiding algorithm that performs matrix multiplications in blocks, such that each block fits within the cache of a GPU, and by careful management of the blocks it minimizes data copying between GPU caches (as data movement is slow). See the page on softmax for details.

An improved version, FlashAttention-2,[79][80][81] was developed to cater to the rising demand for language models capable of handling longer context lengths. It offers enhancements in work partitioning and parallelism, enabling it to achieve up to

230 TFLOPs/s on A100 GPUs (FP16/BF16), a 2x speed increase over the original FlashAttention.

Key advancements in FlashAttention-2 include the reduction of non-matmul FLOPs, improved parallelism over the sequence length dimension, better work partitioning between GPU warps, and added support for head dimensions up to 256 and multi-query attention (MQA) and grouped-query attention (GQA).[82]

Benchmarks revealed FlashAttention-2 to be up to 2x faster than FlashAttention and up to 9x faster than a standard attention implementation in PyTorch. Future developments include optimization for new hardware like H100 GPUs and new data types like FP8.

Multi-Query Attention [edit]

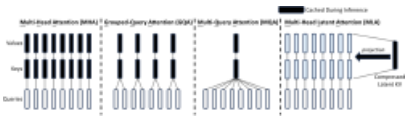
Multi-Query Attention changes the multiheaded attention mechanism.[83] Whereas normally,

$$\text{MultiheadedAttention}(Q,K,V) = \text{Concat}_{i\in[n_{\text{heads}}]}\left(\text{Attention}(XW_i^Q, XW_i^K, XW_i^V)\right)W^O$$

with Multi-Query Attention, there is just one W^K, W^V, thus:

$$\text{MultiQueryAttention}(Q,K,V) = \text{Concat}_{i\in[n_{\text{heads}}]}\left(\text{Attention}(XW_i^Q, XW^K, XW^V)\right)W^O$$

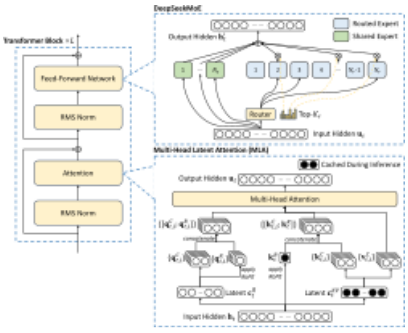
This has a neutral effect on model quality and training speed, but increases inference speed.



Comparison between several different forms of attention mechanism and the amount of KV caching necessary for each

More generally, grouped-query attention (GQA) partitions attention heads into groups, each of which shares the key-value pair. MQA is GQA with one group, while standard multiheaded attention is GQA with the maximal number of groups.[84]

Multihead Latent Attention (MLA) is a low-rank approximation to standard MHA. Specifically, each hidden vector, before entering the attention mechanism, is first projected to two low-dimensional spaces ("latent space"), one for query and one for key-value (KV vector). This design minimizes the KV cache, as only the low-dimensional KV vector needs to be cached.[85]



The architecture of V2, showing both MLA and a variant of mixture of experts[85]:Figure 2

Speculative decoding [edit]

Speculative decoding[86][87] is a method to accelerate token decoding. Similarly to speculative execution in CPUs, future tokens are computed quickly, then verified. If the quickly computed tokens are incorrect, they are discarded and computed slowly.

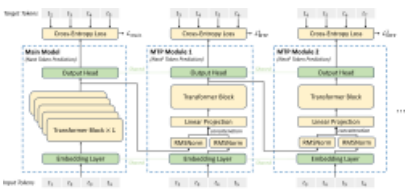
The key factor in speculative decoding is that a Transformer decoder can verify faster than it can decode, in the following sense.

Suppose we have two transformer models like GPT-3 and GPT-3-small, both with a context window size of 512. To generate an entire context window autoregressively with greedy decoding with GPT-3, it must be run for 512 times, each time generating a token , taking time $512T_{\text{GPT-3}}$. However, if we had some educated guess for the values of these tokens, we could verify all of them in parallel, in one run of the model, by checking that each x_t is indeed the token with the largest log-likelihood in the -th output.

In speculative decoding, a smaller model or some other simple heuristic is used to generate a few speculative tokens that are subsequently verified by the larger model. For example, suppose we use GPT-3-small to generate four speculative tokens: $\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \tilde{x}_4$. This only takes $4T_{\text{GPT-3-small}}$. These tokens are then run through the larger GPT-3 in one go. Suppose that $\tilde{x}_1$ and $\tilde{x}_2$ are verified by GPT-3 as what it would have picked, then those are kept, but $\tilde{x}_3$ is not, so $\tilde{x}_3, \tilde{x}_4$ are discarded, and GPT-3 is run on those. This would take $4T_{\text{GPT-3-small}} + 3T_{\text{GPT-3}}$, which might be shorter than $4T_{\text{GPT-3}}$.

For non-greedy decoding, similar ideas apply, except the speculative tokens are accepted or rejected stochastically, in a way that guarantees the final output distribution is the same as if speculative decoding was not used.[86][88]

In Multi-Token Prediction, a single forward pass creates a final embedding vector, which then is un-embedded into a token probability. However, that vector can then be further processed by another Transformer block to predict the *next* token, and so on for arbitrarily many steps into the future. This trades off accuracy for speed, since each new token costs just one more Transformer block, rather than the entire stack.[89][90]



Multi-token prediction

Sub-quadratic transformers [edit]

Training transformer-based architectures can be expensive, especially for long inputs.[91] Many methods have been developed to attempt to address the issue. In the image domain, Swin Transformer is an efficient architecture that performs

attention inside shifting windows.[92] In the audio domain, SepTr decouples the attention in time and frequency domains.[93] *Long Range Arena* (2020)[94] is a standard benchmark for comparing the behavior of transformer architectures over long inputs.

Alternative attention graphs [edit]

The standard attention graph is either all-to-all or causal, both of which scales as $O(N^2)$ where N is the number of tokens in a sequence.

Reformer (2020)[91][95] reduces the computational load from $O(N^2)$ to $O(N \ln N)$ by using locality-sensitive hashing and reversible layers.[96]

Sparse attention[97] uses attention graphs that grows slower than $O(N^2)$. For example, BigBird (2020)[98] uses random small-world networks which grows as $O(N)$.

Ordinary transformers require a memory size that is quadratic in the size of the context window. Attention-free transformers[99] reduce this to a linear dependence while still retaining the advantages of a transformer by linking the key to the value.

Random Feature Attention [edit]

Random Feature Attention (2021)[100] uses Fourier random features:

$$\varphi(x) = \frac{1}{\sqrt{D}}[\cos\langle w_1, x\rangle, \sin\langle w_1, x\rangle, \cdots \cos\langle w_D, x\rangle, \sin\langle w_D, x\rangle]^T$$

where $w_1, \dots, w_D$ are independent samples from the normal distribution $N(0, \sigma^2 I)$. This choice of parameters satisfy $\mathbb{E}[\langle \varphi(x), \varphi(y)\rangle] = e^{-\frac{\|x-y\|^2}{2\sigma^2}}$, or

$$e^{\langle x, y\rangle/\sigma^2} = \mathbb{E}[\langle e^{\|x\|^2/2\sigma^2}\varphi(x), e^{\|y\|^2/2\sigma^2}\varphi(y)\rangle] \approx \langle e^{\|x\|^2/2\sigma^2}\varphi(x), e^{\|y\|^2/2\sigma^2}\varphi(y)\rangle$$

Consequently, the one-headed attention, with one query, can be written as

$$\text{Attention}(q, K, V) = \text{softmax}\left(\frac{qK^T}{\sqrt{d_k}}\right)V \approx \frac{\varphi(q)^T \sum_i e^{\|k_i\|^2/2\sigma^2}\varphi(k_i)v_i^T}{\varphi(q)^T \sum_i e^{\|k_i\|^2/2\sigma^2}\varphi(k_i)}$$

where $\sigma = d_K^{1/4}$. Similarly for multiple queries, and for multiheaded attention.

This approximation can be computed in linear time, as we can compute the matrix $\varphi(k_i)v_i^T$ first, then multiply it with the query. In essence, we have managed to obtain a more precise version of

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

Performer (2022)[101] uses the same Random Feature Attention, but $w_1, \dots, w_D$ are first independently sampled from the normal distribution $N(0, \sigma^2 I)$, then they are Gram-Schmidt processed.

Multimodality [edit]

Transformers can also be used/adapted for modalities (input or output) beyond just text, usually by finding a way to "tokenize" the modality.

Multimodal models can either be trained from scratch, or by finetuning. A 2022 study found that Transformers pretrained only on natural language can be finetuned on only 0.03% of parameters and become competitive with LSTMs on a variety of logical and visual tasks, demonstrating transfer learning.[102] The LLaVA was a vision-language model composed of a language model (Vicuna-13B)[103] and a vision model (ViT-L/14), connected by a linear layer. Only the linear layer is finetuned.[104]

Vision transformers[41] adapt the transformer to computer vision by breaking down input images as a series of patches, turning them into vectors, and treating them like tokens in a standard transformer.

Conformer[42] and later Whisper[105] follow the same pattern for speech recognition, first turning the speech signal into a spectrogram, which is then treated like an image, i.e. broken down into a series of patches, turned into vectors and treated like tokens in a standard transformer.

Perceivers[106][107] are a variant of Transformers designed for multimodality.

For image generation, notable architectures are DALL-E 1 (2021), Parti (2022),[108] Phenaki (2023),[109] and Muse (2023).[110] Unlike later models, DALL-E is not a diffusion model. Instead, it uses a decoder-only Transformer that autoregressively generates a text, followed by the token representation of an image, which is then converted by a variational autoencoder to an image.[111] Parti is an encoder-decoder Transformer, where the encoder processes a text prompt, and the decoder generates a token representation of an image.[112] Muse is an encoder-only Transformer that is trained to

predict masked image tokens from unmasked image tokens. During generation, all input tokens are masked, and the highest-confidence predictions are included for the next iteration, until all tokens are predicted.[110] Phenaki is a text-to-video model. It is a bidirectional masked transformer conditioned on pre-computed text tokens. The generated tokens are then decoded to a video.[109]

Applications [edit]

The transformer has had great success in natural language processing (NLP). Many large language models such as GPT-2, GPT-3, GPT-4, Gemini, AlbertAGPT, Claude, BERT, Grok, XLNet, RoBERTa and ChatGPT demonstrate the ability of transformers to perform a wide variety of NLP-related subtasks and their related real-world applications, including:

- machine translation
- time series prediction
- document summarization
- document generation
- named entity recognition (NER)[113]
- writing computer code based on requirements expressed in natural language.
- speech-to-text

Beyond traditional NLP, the transformer architecture has had success in other applications, such as:

- biological sequence analysis
- video understanding
- protein folding (such as AlphaFold)
- evaluating chess board positions. Using static evaluation alone (that is, with no Minimax search) transformer achieved an Elo of 2895, putting it at grandmaster level.[10]

See also [edit]

- seq2seq – Family of machine learning approaches
- Perceiver – Variant of Transformer designed for multimodal data
- Vision transformer – Machine learning model for vision processing
- Large language model – Type of machine learning model
- BERT (language model) – Series of language models developed by Google AI
- Generative pre-trained transformer – Type of large language model
- T5 (language model) – Series of large language models developed by Google AI

Notes [edit]

1. ^ Gated recurrent units (2014) further reduced its complexity.
2. ^ Some architectures, such as RWKV or state space models, avoid the issue.

References [edit]

1. ^ a b c d e f g h i j k l Vaswani, Ashish; Shazeer, Noam; Parmar, Niki; Uszkoreit, Jakob; Jones, Llion; Gomez, Aidan N; Kaiser, Łukasz; Polosukhin, Illia (2017). "Attention is All you Need" (PDF). *Advances in Neural Information Processing Systems*. **30**. Curran Associates, Inc.
2. ^ Hochreiter, Sepp; Schmidhuber, Jürgen (1 November 1997). "Long Short-Term Memory". *Neural Computation*. **9** (8): 1735–1780. doi:10.1162/neco.1997.9.8.1735. ISSN 0899-7667. PMID 9377276. S2CID 1915014.
3. ^ a b "Better Language Models and Their Implications". *OpenAI*. 2019-02-14. Archived from the original on 2020-12-19. Retrieved 2019-08-25.
4. ^ a b Bahdanau; Cho, Kyunghyun; Bengio, Yoshua (September 1, 2014). "Neural Machine Translation by Jointly Learning to Align and Translate". arXiv:1409.0473 [cs.CL].
5. ^ Luong, Minh-Thang; Pham, Hieu; Manning, Christopher D. (August 17, 2015). "Effective Approaches to Attention-based Neural Machine Translation". arXiv:1508.04025 [cs.CL].
6. ^ a b Chen, Lili; Lu, Kevin; Rajeswaran, Aravind; Lee, Kimin; Grover, Aditya; Laskin, Michael; Abbeel, Pieter; Srinivas, Aravind; Mordatch, Igor (2021-06-24), *Decision Transformer: Reinforcement Learning via Sequence Modeling*, arXiv:2106.01345
7. ^ Parisotto, Emilio; Song, Francis; Rae, Jack; Pascanu, Razvan; Gulcehre, Caglar; Jayakumar, Siddhant; Jaderberg, Max; Kaufman, Raphaël Lopez; Clark, Aidan; Noury, Seb; Botvinick, Matthew; Heess, Nicolas; Hadsell, Raia (2020-11-21). "Stabilizing Transformers for Reinforcement Learning". *Proceedings of the 37th International Conference on Machine Learning*. PMLR: 7487–7498.
8. ^ Radford, Alec; Jong Wook Kim; Xu, Tao; Brockman, Greg; McLeavey, Christine; Sutskever, Ilya (2022). "Robust Speech Recognition via Large-Scale Weak Supervision". arXiv:2212.04356 [eess.AS].
9. ^ Monastirsky, Maxim; Azulay, Osher; Sintov, Avishai (February 2023). "Learning to Throw With a Handful of Samples Using Decision Transformers". *IEEE Robotics and Automation Letters*. **8** (2): 576–583. doi:10.1109/LRA.2022.3229266. ISSN 2377-3766.
10. ^ a b Ruoss, Anian; Delétang, Grégoire; Medapati, Sourabh; Grau-Moya, Jordi; Wenliang, Li; Catt, Elliot; Reid, John; Genewein, Tim (2024-02-07). "Grandmaster-Level Chess Without Search". arXiv:2402.04494v1 [cs.LG].

11. ^ a b Wolf, Thomas; Debut, Lysandre; Sanh, Victor; Chaumond, Julien; Delangue, Clement; Moi, Anthony; Cistac, Pierric; Rault, Tim; Louf, Remi; Funtowicz, Morgan; Davison, Joe; Shleifer, Sam; von Platen, Patrick; Ma, Clara; Jernite, Yacine; Plu, Julien; Xu, Canwen; Le Scao, Teven; Gugger, Sylvain; Drame, Mariama; Lhoest, Quentin; Rush, Alexander (2020). "Transformers: State-of-the-Art Natural Language Processing". *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. pp. 38–45. doi:10.18653/v1/2020.emnlp-demos.6. S2CID 208117506.
12. ^ a b c "Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing". *Google AI Blog*. 2 November 2018. Archived from the original on 2021-01-13. Retrieved 2019-08-25.
13. ^ Feldman, J. A.; Ballard, D. H. (1982-07-01). "Connectionist models and their properties". *Cognitive Science*. **6** (3): 205–254. doi:10.1016/S0364-0213(82)80001-3. ISSN 0364-0213.
14. ^ Rumelhart, David E.; McClelland, James L.; Hinton, Geoffrey E. (1987-07-29). *Parallel Distributed Processing, Volume 1: Explorations in the Microstructure of Cognition: Foundations, Chapter 2* (PDF). Cambridge, Mass: Bradford Books. ISBN 978-0-262-68053-0.
15. ^ Giles, C. Lee; Maxwell, Tom (1987-12-01). "Learning, invariance, and generalization in high-order neural networks". *Applied Optics*. **26** (23): 4972–4978. doi:10.1364/AO.26.004972. ISSN 0003-6935. PMID 20523475.
16. ^ a b Schmidhuber, Jürgen (1992). "Learning to control fast-weight memories: an alternative to recurrent nets" (PDF). *Neural Computation*. **4** (1): 131–139. doi:10.1162/neco.1992.4.1.131. S2CID 16683347.
17. ^ Christoph von der Malsburg: The correlation theory of brain function. Internal Report 81-2, MPI Biophysical Chemistry, 1981. http://cogprints.org/1380/1/vdM_correlation.pdf See Reprint in Models of Neural Networks II, chapter 2, pages 95–119. Springer, Berlin, 1994.
18. ^ Jerome A. Feldman, "Dynamic connections in neural networks," Biological Cybernetics, vol. 46, no. 1, pp. 27–39, Dec. 1982.
19. ^ Hinton, Geoffrey E.; Plaut, David C. (1987). "Using Fast Weights to Deblur Old Memories". *Proceedings of the Annual Meeting of the Cognitive Science Society*. **9**.
20. ^ Katharopoulos, Angelos; Vyas, Apoorv; Pappas, Nikolaos; Fleuret, François (2020). "Transformers are RNNs: Fast autoregressive Transformers with linear attention". *ICML 2020*. PMLR. pp. 5156–5165.
21. ^ Schlag, Imanol; Irie, Kazuki; Schmidhuber, Jürgen (2021). "Linear Transformers Are Secretly Fast Weight Programmers". *ICML 2021*. Springer. pp. 9355–9366.
22. ^ a b Cho, Kyunghyun; van Merriënboer, Bart; Gulcehre, Caglar; Bahdanau, Dzmitry; Bougares, Fethi; Schwenk, Holger; Bengio, Yoshua (October 2014). "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In Moschitti, Alessandro; Pang, Bo; Daelemans, Walter (eds.). *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics. pp. 1724–1734. arXiv:1406.1078. doi:10.3115/v1/D14-1179.
23. ^ a b Sutskever, Ilya; Vinyals, Oriol; Le, Quoc Viet (14 Dec 2014). "Sequence to sequence learning with neural networks". arXiv:1409.3215 [cs.CL]. [first version posted to arXiv on 10 Sep 2014]
24. ^ Chung, Junyoung; Gulcehre, Caglar; Cho, KyungHyun; Bengio, Yoshua (2014). "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". arXiv:1412.3555 [cs.NE].
25. ^ Gruber, N.; Jockisch, A. (2020), "Are GRU cells more specific and LSTM cells more sensitive in motive classification of text?", *Frontiers in Artificial Intelligence*, **3**: 40, doi:10.3389/frai.2020.00040, PMC 7861254, PMID 33733157, S2CID 220252321
26. ^ Sutskever, Ilya; Vinyals, Oriol; Le, Quoc V (2014). "Sequence to Sequence Learning with Neural Networks". *Advances in Neural Information Processing Systems*. **27**. Curran Associates, Inc. arXiv:1409.3215.
27. ^ Luong, Minh-Thang; Pham, Hieu; Manning, Christopher D. (2015). "Effective Approaches to Attention-based Neural Machine Translation". arXiv:1508.04025 [cs.CL].
28. ^ Wu, Yonghui; et al. (2016-09-01). "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". arXiv:1609.08144 [cs.CL].
29. ^ Lewis-Kraus, Gideon (2016-12-14). "The Great A.I. Awakening". *The New York Times*. ISSN 0362-4331. Archived from the original on 24 May 2023. Retrieved 2023-06-22.
30. ^ Parikh, Ankur P.; Täckström, Oscar; Das, Dipanjan; Uszkoreit, Jakob (2016-09-25). "A Decomposable Attention Model for Natural Language Inference". arXiv:1606.01933 [cs.CL].
31. ^ a b Levy, Steven. "8 Google Employees Invented Modern AI. Here's the Inside Story". *Wired*. ISSN 1059-1028. Archived from the original on 20 Mar 2024. Retrieved 2024-08-06.
32. ^ Cheng, Jianpeng; Dong, Li; Lapata, Mirella (November 2016). "Long Short-Term Memory-Networks for Machine Reading". In Su, Jian; Duh, Kevin; Carreras, Xavier (eds.). *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics. pp. 551–561. doi:10.18653/v1/D16-1053.
33. ^ Peng, Bo; Alcaide, Eric; Anthony, Quentin; Albalak, Alon; Arcadinho, Samuel; Biderman, Stella; Cao, Huanqi; Cheng, Xin; Chung, Michael (2023-12-10), *RWKV: Reinventing RNNs for the Transformer Era*, arXiv:2305.13048
34. ^ Marche, Stephen (2024-08-23). "Was Linguistic A.I. Created by Accident?". *The New Yorker*. ISSN 0028-792X. Retrieved 2024-08-27.
35. ^ a b c d e f Devlin, Jacob; Chang, Ming-Wei; Lee, Kenton; Toutanova, Kristina (11 October 2018). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". arXiv:1810.04805v2 [cs.CL].
36. ^ "Google: BERT now used on almost every English query". *Search Engine Land*. 2020-10-15. Retrieved 2020-11-24.
37. ^ "Recent Advances in Google Translate". *research.google*. Retrieved 2024-05-08.
38. ^ "The inside story of how ChatGPT was built from the people who made it". *MIT Technology Review*. Retrieved 2024-08-06.
39. ^ "Improving language understanding with unsupervised learning". *openai.com*. June 11, 2018. Archived from the original on 2023-03-18. Retrieved 2023-03-18.
40. ^ *finetune-transformer-lm*, OpenAI, June 11, 2018, retrieved 2023-05-01
41. ^ a b Dosovitskiy, Alexey; Beyer, Lucas; Kolesnikov, Alexander; Weissenborn, Dirk; Zhai, Xiaohua; Unterthiner, Thomas; Dehghani, Mostafa; Minderer, Matthias; Heigold, Georg; Gelly, Sylvain; Uszkoreit, Jakob (2021-06-03). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". arXiv:2010.11929 [cs.CV].
42. ^ a b Gulati, Anmol; Qin, James; Chiu, Chung-Cheng; Parmar, Niki; Zhang, Yu; Yu, Jiahui; Han, Wei; Wang, Shibo; Zhang, Zhengdong; Wu, Yonghui; Pang, Ruoming (2020). "Conformer: Convolution-augmented Transformer for Speech Recognition". arXiv:2005.08100 [eess.AS].
43. ^ Choromanski, Krzysztof; Likhosherstov, Valerii; Dohan, David; Song, Xingyou; Gane, Andreea; Sarlos, Tamas; Hawkins, Peter; Davis, Jared; Mohiuddin, Afroz (2022-11-19), *Rethinking Attention with Performers*, arXiv:2009.14794
44. ^ Liu, Zhuang; Mao, Hanzi; Wu, Chao-Yuan; Feichtenhofer, Christoph; Darrell, Trevor; Xie, Saining (2022). *A ConvNet for the 2020s*. Conference on Computer Vision and Pattern Recognition. pp. 11976–11986.
45. ^ Esser, Patrick; Kulal, Sumith; Blattmann, Andreas; Entezari, Rahim; Müller, Jonas; Saini, Harry; Levi, Yam; Lorenz, Dominik; Sauer, Axel (2024-03-05), *Scaling Rectified Flow Transformers for High-Resolution Image Synthesis*, arXiv:2403.03206

46. ^ a b Xiong, Ruibin; Yang, Yunchang; He, Di; Zheng, Kai; Zheng, Shuxin; Xing, Chen; Zhang, Huishuai; Lan, Yanyan; Wang, Liwei; Liu, Tie-Yan (2020-06-29). "On Layer Normalization in the Transformer Architecture". arXiv:2002.04745 [cs.LG].
47. ^ Raffel, Colin; Shazeer, Noam; Roberts, Adam; Lee, Katherine; Narang, Sharan; Matena, Michael; Zhou, Yanqi; Li, Wei; Liu, Peter J. (2020-01-01). "Exploring the limits of transfer learning with a unified text-to-text transformer". *The Journal of Machine Learning Research*. **21** (1): 140:5485–140:5551. arXiv:1910.10683. ISSN 1532-4435.
48. ^ Raffel, Colin; Shazeer, Noam; Roberts, Adam; Lee, Katherine; Narang, Sharan; Matena, Michael; Zhou, Yanqi; Li, Wei; Liu, Peter J. (2019). "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". arXiv:1910.10683 [cs.LG].
49. ^ a b "Masked language modeling". *huggingface.co*. Retrieved 2023-10-05.
50. ^ a b "Causal language modeling". *huggingface.co*. Retrieved 2023-10-05.
51. ^ a b c d Tay, Yi; Dehghani, Mostafa; Tran, Vinh Q.; Garcia, Xavier; Wei, Jason; Wang, Xuezhi; Chung, Hyung Won; Shakeri, Siamak; Bahri, Dara (2023-02-28), *UL2: Unifying Language Learning Paradigms*, arXiv:2205.05131
52. ^ Press, Ofir; Wolf, Lior (2017-02-21), *Using the Output Embedding to Improve Language Models*, arXiv:1608.05859
53. ^ Lintz, Nathan (2016-04-18). "Sequence Modeling with Neural Networks (Part 2): Attention Models". *Indico*. Archived from the original on 2020-10-21. Retrieved 2019-10-15.
54. ^ a b c Alammar, Jay. "The Illustrated Transformer". *jalammar.github.io*. Archived from the original on 2020-10-18. Retrieved 2019-10-15.
55. ^ Team, Keras. "Keras documentation: GPT2Backbone model". *keras.io*. Retrieved 2024-08-08.
56. ^ Clark, Kevin; Khandelwal, Urvashi; Levy, Omer; Manning, Christopher D. (August 2019). "What Does BERT Look at? An Analysis of BERT's Attention". *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Florence, Italy: Association for Computational Linguistics: 276–286. arXiv:1906.04341. doi:10.18653/v1/W19-4828. Archived from the original on 2020-10-21. Retrieved 2020-05-20.
57. ^ Yang, Zhilin; Dai, Zihang; Yang, Yiming; Carbonell, Jaime; Salakhutdinov, Russ R; Le, Quoc V (2019). "XLNet: Generalized Autoregressive Pretraining for Language Understanding". *Advances in Neural Information Processing Systems*. **32**. Curran Associates, Inc. arXiv:1906.08237.
58. ^ Radford, Alec; Narasimhan, Karthik; Salimans, Tim; Sutskever, Ilya (11 June 2018). "Improving Language Understanding by Generative Pre-Training" (PDF). OpenAI. p. 12. Archived (PDF) from the original on 26 January 2021. Retrieved 23 January 2021.
59. ^ Wang, Qiang; Li, Bei; Xiao, Tong; Zhu, Jingbo; Li, Changliang; Wong, Derek F.; Chao, Lidia S. (2019-06-04), *Learning Deep Transformer Models for Machine Translation*, arXiv:1906.01787
60. ^ Phuong, Mary; Hutter, Marcus (2022-07-19), *Formal Algorithms for Transformers*, arXiv:2207.09238
61. ^ a b c Raffel, Colin; Shazeer, Noam; Roberts, Adam; Lee, Katherine; Narang, Sharan; Matena, Michael; Zhou, Yanqi; Li, Wei; Liu, Peter J. (2020). "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". *Journal of Machine Learning Research*. **21** (140): 1–67. arXiv:1910.10683. ISSN 1533-7928.
62. ^ "Recent Advances in Google Translate". *Google Research*. June 8, 2020. Archived from the original on 4 Jul 2024. Retrieved 2024-08-07.
63. ^ a b Shazeer, Noam (2020-02-01). "GLU Variants Improve Transformer". arXiv:2002.05202 [cs.LG].
64. ^ Hendrycks, Dan; Gimpel, Kevin (2016-06-27). "Gaussian Error Linear Units (GELUs)". arXiv:1606.08415v5 [cs.LG].
65. ^ Zhang, Biao; Sennrich, Rico (2019). "Root Mean Square Layer Normalization". *Advances in Neural Information Processing Systems*. **32**. Curran Associates, Inc. arXiv:1910.07467.
66. ^ Tembine, Hamidou, Manzoor Ahmed Khan, and Issa Bamia. 2024. "Mean-Field-Type Transformers" Mathematics 12, no. 22: 3506. https://doi.org/10.3390/math12223506
67. ^ a b Nguyen, Toan Q.; Salazar, Julian (2019-11-02). Niehues, Jan; Cattoni, Rolando; Stüker, Sebastian; Negri, Matteo; Turchi, Marco; Ha, Thanh-Le; Salesky, Elizabeth; Sanabria, Ramon; Barrault, Loic (eds.). "Transformers without Tears: Improving the Normalization of Self-Attention". *Proceedings of the 16th International Conference on Spoken Language Translation*. Hong Kong: Association for Computational Linguistics. arXiv:1910.05895. doi:10.5281/zenodo.3525484.
68. ^ Dufter, Philipp; Schmitt, Martin; Schütze, Hinrich (2022-06-06). "Position Information in Transformers: An Overview". *Computational Linguistics*. **48** (3): 733–763. arXiv:2102.11090. doi:10.1162/coli_a_00445. ISSN 0891-2017. S2CID 231986066.
69. ^ Gehring, Jonas; Auli, Michael; Grangier, David; Yarats, Denis; Dauphin, Yann N. (2017-07-17). "Convolutional Sequence to Sequence Learning". *Proceedings of the 34th International Conference on Machine Learning*. PMLR: 1243–1252.
70. ^ Haviv, Adi; Ram, Ori; Press, Ofir; Izsak, Peter; Levy, Omer (2022-12-05), *Transformer Language Models without Positional Encodings Still Learn Positional Information*, arXiv:2203.16634
71. ^ Su, Jianlin; Lu, Yu; Pan, Shengfeng; Murtadha, Ahmed; Wen, Bo; Liu, Yunfeng (2021-04-01). "RoFormer: Enhanced Transformer with Rotary Position Embedding". arXiv:2104.09864 [cs.CL].
72. ^ Press, Ofir; Smith, Noah A.; Lewis, Mike (2021-08-01). "Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation". arXiv:2108.12409 [cs.CL].
73. ^ Shaw, Peter; Uszkoreit, Jakob; Vaswani, Ashish (2018). "Self-Attention with Relative Position Representations". arXiv:1803.02155 [cs.CL].
74. ^ Ke, Guolin; He, Di; Liu, Tie-Yan (2021-03-15), *Rethinking Positional Encoding in Language Pre-training*, arXiv:2006.15595
75. ^ Kwon, Woosuk; Li, Zhuohan; Zhuang, Siyuan; Sheng, Ying; Zheng, Lianmin; Yu, Cody Hao; Gonzalez, Joseph; Zhang, Hao; Stoica, Ion (2023-10-23). "Efficient Memory Management for Large Language Model Serving with PagedAttention". *Proceedings of the 29th Symposium on Operating Systems Principles*. SOSP '23. New York, NY, USA: Association for Computing Machinery. pp. 611–626. arXiv:2309.06180. doi:10.1145/3600006.3613165. ISBN 979-8-4007-0229-7.
76. ^ *vllm-project/vllm*, vLLM, 2024-06-20, retrieved 2024-06-20
77. ^ Contribution), Woosuk Kwon* (Equal, Zhuohan Li* (Equal, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Yu, Joey Gonzalez, Hao Zhang, and Ion Stoica (* Equal (2023-06-20). "vLLM: Easy, Fast, and Cheap LLM Serving with PagedAttention". *vLLM Blog*. Retrieved 2024-06-20.
78. ^ Dao, Tri; Fu, Dan; Ermon, Stefano; Rudra, Atri; Ré, Christopher (2022-12-06). "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness". *Advances in Neural Information Processing Systems*. **35**: 16344–16359. arXiv:2205.14135.
79. ^ "Stanford CRFM". *crfm.stanford.edu*. Retrieved 2023-07-18.
80. ^ "FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning". *Princeton NLP*. 2023-06-17. Retrieved 2023-07-18.
81. ^ "Introducing Together AI Chief Scientist Tri Dao, as he releases FlashAttention-2 to speed up model training and inference". *TOGETHER*. Retrieved 2023-07-18.
82. ^ Ainslie, Joshua; Lee-Thorp, James; de Jong, Michiel; Zemlyanskiy, Yury; Lebrón, Federico; Sanghai, Sumit (2023-12-23). "GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints". arXiv:2305.13245 [cs.CL].

83. ^ Chowdhery, Aakanksha; Narang, Sharan; Devlin, Jacob; Bosma, Maarten; Mishra, Gaurav; Roberts, Adam; Barham, Paul; Chung, Hyung Won; Sutton, Charles; Gehrmann, Sebastian; Schuh, Parker; Shi, Kensen; Tsvyashchenko, Sasha; Maynez, Joshua; Rao, Abhishek (2022-04-01). "PaLM: Scaling Language Modeling with Pathways". arXiv:2204.02311 [cs.CL].
84. ^ Ainslie, Joshua; Lee-Thorp, James; de Jong, Michiel; Zemlyanskiy, Yury; Lebrón, Federico; Sanghai, Sumit (2023-12-23), *GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints*, arXiv:2305.13245
85. ^ a b DeepSeek-AI; Liu, Aixin; Feng, Bei; Wang, Bin; Wang, Bingxuan; Liu, Bo; Zhao, Chenggang; Dengr, Chengqi; Ruan, Chong (19 June 2024), *DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model*, arXiv:2405.04434.
86. ^ a b Leviathan, Yaniv; Kalman, Matan; Matias, Yossi (2023-05-18), *Fast Inference from Transformers via Speculative Decoding*, arXiv:2211.17192
87. ^ Fu, Yao (2023-12-13). "Towards 100x Speedup: Full Stack Transformer Inference Optimization".
88. ^ Chen, Charlie; Borgeaud, Sebastian; Irving, Geoffrey; Lespiau, Jean-Baptiste; Sifre, Laurent; Jumper, John (2023-02-02), *Accelerating Large Language Model Decoding with Speculative Sampling*, arXiv:2302.01318
89. ^ Gloeckle, Fabian; Badr Youbi Idrissi; Rozière, Baptiste; Lopez-Paz, David; Synnaeve, Gabriel (2024). "Better & Faster Large Language Models via Multi-token Prediction". arXiv:2404.19737 [cs.CL].
90. ^ DeepSeek-AI; et al. (2024). "DeepSeek-V3 Technical Report". arXiv:2412.19437 [cs.CL].
91. ^ a b Kitaev, Nikita; Kaiser, Łukasz; Levskaya, Anselm (2020). "Reformer: The Efficient Transformer". arXiv:2001.04451 [cs.LG].
92. ^ Liu, Ze; Lin, Yutong; Cao, Yue; Hu, Han; Wei, Yixuan; Zhang, Zheng; Lin, Stephen; Guo, Baining (2021). "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows". *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE. pp. 9992–10002. arXiv:2103.14030. doi:10.1109/ICCV48922.2021.00986. ISBN 978-1-6654-2812-5.
93. ^ Ristea, Nicolaea Catalin; Ionescu, Radu Tudor; Khan, Fahad Shahbaz (2022-09-18). "SepTr: Separable Transformer for Audio Spectrogram Processing". *Interspeech*. ISCA: 4103–4107. arXiv:2203.09581. doi:10.21437/Interspeech.2022-249.
94. ^ Tay, Yi; Dehghani, Mostafa; Abnar, Samira; Shen, Yikang; Bahri, Dara; Pham, Philip; Rao, Jinfeng; Yang, Liu; Ruder, Sebastian; Metzler, Donald (2020-11-08). "Long Range Arena: A Benchmark for Efficient Transformers". arXiv:2011.04006 [cs.LG].
95. ^ "Reformer: The Efficient Transformer". *Google AI Blog*. 16 January 2020. Archived from the original on 2020-10-22. Retrieved 2020-10-22.
96. ^ Gomez, Aidan N; Ren, Mengye; Urtasun, Raquel; Grosse, Roger B (2017). "The Reversible Residual Network: Backpropagation Without Storing Activations". *Advances in Neural Information Processing Systems*. **30**. Curran Associates, Inc. arXiv:1707.04585.
97. ^ Child, Rewon; Gray, Scott; Radford, Alec; Sutskever, Ilya (2019-04-23), *Generating Long Sequences with Sparse Transformers*, arXiv:1904.10509
98. ^ "Constructing Transformers For Longer Sequences with Sparse Attention Methods". *Google AI Blog*. 25 March 2021. Archived from the original on 2021-09-18. Retrieved 2021-05-28.
99. ^ Zhai, Shuangfei; Talbott, Walter; Srivastava, Nitish; Huang, Chen; Goh, Hanlin; Zhang, Ruixiang; Susskind, Josh (2021-09-21). "An Attention Free Transformer". arXiv:2105.14103 [cs.LG].
100. ^ Peng, Hao; Pappas, Nikolaos; Yogatama, Dani; Schwartz, Roy; Smith, Noah A.; Kong, Lingpeng (2021-03-19). "Random Feature Attention". arXiv:2103.02143 [cs.CL].
101. ^ Choromanski, Krzysztof; Likhosherstov, Valerii; Dohan, David; Song, Xingyou; Gane, Andreea; Sarlos, Tamas; Hawkins, Peter; Davis, Jared; Belanger, David; Colwell, Lucy; Weller, Adrian (2020-09-30). "Masked Language Modeling for Proteins via Linearly Scalable Long-Context Transformers". arXiv:2006.03555 [cs.LG].
102. ^ Lu, Kevin; Grover, Aditya; Abbeel, Pieter; Mordatch, Igor (2022-06-28). "Frozen Pretrained Transformers as Universal Computation Engines". *Proceedings of the AAAI Conference on Artificial Intelligence*. **36** (7): 7628–7636. doi:10.1609/aaai.v36i7.20729. ISSN 2374-3468.
103. ^ "Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality | LMSYS Org". *lmsys.org*. Retrieved 2024-08-11.
104. ^ Liu, Haotian; Li, Chunyuan; Wu, Qingyang; Lee, Yong Jae (2023-12-15). "Visual Instruction Tuning". *Advances in Neural Information Processing Systems*. **36**: 34892–34916.
105. ^ Radford, Alec; Kim, Jong Wook; Xu, Tao; Brockman, Greg; McLeavey, Christine; Sutskever, Ilya (2022). "Robust Speech Recognition via Large-Scale Weak Supervision". arXiv:2212.04356 [eess.AS].
106. ^ Jaegle, Andrew; Gimeno, Felix; Brock, Andrew; Zisserman, Andrew; Vinyals, Oriol; Carreira, Joao (2021-06-22). "Perceiver: General Perception with Iterative Attention". arXiv:2103.03206 [cs.CV].
107. ^ Jaegle, Andrew; Borgeaud, Sebastian; Alayrac, Jean-Baptiste; Doersch, Carl; Ionescu, Catalin; Ding, David; Koppula, Skanda; Zoran, Daniel; Brock, Andrew; Shelhamer, Evan; Hénaff, Olivier (2021-08-02). "Perceiver IO: A General Architecture for Structured Inputs & Outputs". arXiv:2107.14795 [cs.LG].
108. ^ "Parti: Pathways Autoregressive Text-to-Image Model". *sites.research.google*. Retrieved 2024-08-09.
109. ^ a b Villegas, Ruben; Babaeizadeh, Mohammad; Kindermans, Pieter-Jan; Moraldo, Hernan; Zhang, Han; Saffar, Mohammad Taghi; Castro, Santiago; Kunze, Julius; Erhan, Dumitru (2022-09-29). "Phenaki: Variable Length Video Generation from Open Domain Textual Descriptions". `{{cite journal}}`: Cite journal requires `|journal=` (help)
110. ^ a b Chang, Huiwen; Zhang, Han; Barber, Jarred; Maschinot, A. J.; Lezama, Jose; Jiang, Lu; Yang, Ming-Hsuan; Murphy, Kevin; Freeman, William T. (2023-01-02). "Muse: Text-To-Image Generation via Masked Generative Transformers". arXiv:2301.00704 [cs.CV].
111. ^ Ramesh, Aditya; Pavlov, Mikhail; Goh, Gabriel; Gray, Scott; Voss, Chelsea; Radford, Alec; Chen, Mark; Sutskever, Ilya (2021-02-26), *Zero-Shot Text-to-Image Generation*, arXiv:2102.12092
112. ^ Yu, Jiahui; Xu, Yuanzhong; Koh, Jing Yu; Luong, Thang; Baid, Gunjan; Wang, Zirui; Vasudevan, Vijay; Ku, Alexander; Yang, Yinfei (2022-06-21), *Scaling Autoregressive Models for Content-Rich Text-to-Image Generation*, arXiv:2206.10789
113. ^ Kariampuzha, William; Alyea, Gioconda; Qu, Sue; Sanjak, Jaleal; Mathé, Ewy; Sid, Eric; Chatelaine, Haley; Yadaw, Arjun; Xu, Yanji; Zhu, Qian (2023). "Precision information extraction for rare disease epidemiology at scale". *Journal of Translational Medicine*. **21** (1): 157. doi:10.1186/s12967-023-04011-y. PMC 9972634. PMID 36855134.

Further reading [edit]

- Alexander Rush, The Annotated transformer Archived 2021-09-22 at the Wayback Machine, Harvard NLP group, 3 April 2018
- Phuong, Mary; Hutter, Marcus (2022). "Formal Algorithms for Transformers". arXiv:2207.09238 [cs.LG].
- Ferrando, Javier; Sarti, Gabriele; Bisazza, Arianna; Costa-jussà, Marta R. (2024-05-01). "A Primer on the Inner Workings of Transformer-based Language Models". arXiv:2405.00208 [cs.CL].

- Leech, Gavin (2024-11-06). "Transformer++". *argmin gravitas*. Archived from the original on 2025-02-26. Retrieved 2025-05-08.

v · t · e

Google AI

Google · Google Brain · Google DeepMind

Computer programs	**AlphaGo**	**Versions**	AlphaGo (2015) · Master (2016) · AlphaGo Zero (2017) · AlphaZero (2017) · MuZero (2019)
		Competitions	Fan Hui (2015) · Lee Sedol (2016) · Ke Jie (2017)
		In popular culture	*AlphaGo* (2017) · *The MANIAC* (2023)
	Other		AlphaFold (2018) · AlphaStar (2019) · AlphaDev (2023) · AlphaGeometry (2024) · AlphaGenome (2025)
Machine learning	**Neural networks**		Inception (2014) · WaveNet (2016) · MobileNet (2017) · **Transformer** (2017) · EfficientNet (2019) · Gato (2022)
	Other		Quantum Artificial Intelligence Lab · TensorFlow · Tensor Processing Unit
Generative AI	**Chatbots**		Assistant (2016) · Sparrow (2022) · Gemini (2023)
	Models		BERT (2018) · XLNet (2019) · T5 (2019) · LaMDA (2021) · Chinchilla (2022) · PaLM (2022) · Imagen (2023) · Gemini (2023) · VideoPoet (2024) · Veo (text-to-video model) (2024)
	Other		DreamBooth (2022) · NotebookLM (2023) · Vids (2024) · Gemini Robotics (2025)
See also			"Attention Is All You Need" · Future of Go Summit · Generative pre-trained transformer · Google Labs · Google Pixel · Google Workspace · Robot Constitution

Category · Commons

v · t · e

Artificial intelligence (AI)

History (timeline)

Concepts		Parameter (Hyperparameter) · Loss functions · Regression (Bias–variance tradeoff · Double descent · Overfitting) · Clustering · Gradient descent (SGD · Quasi-Newton method · Conjugate gradient method) · Backpropagation · Attention · Convolution · Normalization (Batchnorm) · Activation (Softmax · Sigmoid · Rectifier) · Gating · Weight initialization · Regularization · Datasets (Augmentation) · Prompt engineering · Reinforcement learning (Q-learning · SARSA · Imitation · Policy gradient) · Diffusion · Latent diffusion model · Autoregression · Adversary · RAG · Uncanny valley · RLHF · Self-supervised learning · Reflection · Recursive self-improvement · Hallucination · Word embedding · Vibe coding
Applications		Machine learning (In-context learning) · Artificial neural network (Deep learning) · Language model (Large language model · NMT) · Reasoning language model · Model Context Protocol · Intelligent agent · Artificial human companion · Humanity's Last Exam · Artificial general intelligence (AGI)
Implementations	**Audio–visual**	AlexNet · WaveNet · Human image synthesis · HWR · OCR · Speech synthesis (15.ai · ElevenLabs) · Speech recognition (Whisper) · Facial recognition · AlphaFold · Text-to-image models (Aurora · DALL-E · Firefly · Flux · Ideogram · Imagen · Midjourney · Recraft · Stable Diffusion) · Text-to-video models (Dream Machine · Runway Gen · Hailuo AI · Kling · Sora · Veo) · Music generation (Suno AI · Udio)
	Text	Word2vec · Seq2seq · GloVe · BERT · T5 · Llama · Chinchilla AI · PaLM · GPT (1 · 2 · 3 · J · ChatGPT · 4 · 4o · o1 · o3 · 4.5 · 4.1 · o4-mini) · Claude · Gemini (chatbot) · Grok · LaMDA · BLOOM · Project Debater · IBM Watson · IBM Watsonx · Granite · PanGu-Σ · DeepSeek · Qwen
	Decisional	AlphaGo · AlphaZero · OpenAI Five · Self-driving car · MuZero · Action selection (AutoGPT) · Robot control
People		Alan Turing · Warren Sturgis McCulloch · Walter Pitts · John von Neumann · Claude Shannon · Marvin Minsky · John McCarthy · Nathaniel Rochester · Allen Newell · Cliff Shaw · Herbert A. Simon · Oliver Selfridge · Frank Rosenblatt · Bernard Widrow · Joseph Weizenbaum · Seymour Papert · Seppo Linnainmaa · Paul Werbos · Jürgen Schmidhuber · Yann LeCun · Geoffrey Hinton · John Hopfield · Yoshua Bengio · Lotfi A. Zadeh · Stephen Grossberg · Alex Graves · James Goodnight · Andrew Ng · Fei-Fei Li · Ilya Sutskever · Alex Krizhevsky · Ian Goodfellow · Demis Hassabis · David Silver · Andrej Karpathy · Ashish Vaswani · Noam Shazeer · Aidan Gomez

Architectures	Neural Turing machine · Differentiable neural computer · **Transformer** (Vision transformer (ViT)) · Recurrent neural network (RNN) · Long short-term memory (LSTM) · Gated recurrent unit (GRU) · Echo state network · Multilayer perceptron (MLP) · Convolutional neural network (CNN) · Residual neural network (RNN) · Highway network · Mamba · Autoencoder · Variational autoencoder (VAE) · Generative adversarial network (GAN) · Graph neural network (GNN)
Portals (Technology) · Category (Artificial neural networks · Machine learning) · List (Companies · Projects)	

Categories: Google software Neural network architectures 2017 in artificial intelligence

This page was last edited on 26 June 2025, at 19:01 (UTC).

Text is available under the Creative Commons Attribution-ShareAlike 4.0 License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.

Privacy policy About Wikipedia Disclaimers Contact Wikipedia Code of Conduct Developers Statistics Cookie statement Mobile view