Attention Mechanisms

Bora Deren
437783
Bachelor Computer Science
RWTH Aachen
bora.deren@rwth-aachen.de

***Abstract*—This paper introduces attention mechanisms and the Transformer architecture. Starting from earlier sequence models such as n-grams, RNNs, and LSTMs, we explain why self-attention became the dominant approach for self-supervised language modeling. We describe the core building blocks of Transformers, introduce their variants, and compare them with RNNs on a small character-level benchmark, and discuss their strengths, limitations, and applications.**

I. Introduction

Understanding human language, and thus language modeling, is often described as an AI-complete problem for machines. Besides communication, language is also a tool for abstraction, reasoning, and social coordination. Therefore, progress in language understanding is frequently thought of as a progress towards artificial general intelligence (AGI).

From a philosophical perspective, language modeling is more complex than just an engineering problem. What does it mean for a system to "understand" a sentence rather than just predicting the next word? Can meaning be reduced to statistical patterns in text, or does it require foundations in perception and action? Machine learning and natural language processing have shown that scalable models trained on large corpora can capture many useful linguistic patterns.

Since the rise of digital computers, a range of methods has been proposed for language modeling. Among the earliest approaches were n-gram models, which estimate the probability of a token given a limited context. Later, neural architectures such as recurrent neural networks (RNNs), variants such as LSTMs, and convolutional neural networks (CNNs) became widely used. These models can process tokens sequentially, but they have several limitations, particularly when modeling long-range dependencies and enabling parallel computation.

To address these issues, attention mechanisms, inspired in part by biological attention, were introduced to augment neural language models. The Transformer architecture proposed in [1] adopted attention as a core design component. This architecture was quickly adopted beyond natural language processing (NLP), as researchers in other domains adapted its core ideas. Today, Transformers are used in a wide variety of fields, including computer vision and graph representation learning.

In this paper, we examine the development of neural language modeling and analyze why "Attention Is All You Need". After providing historical background and discussing earlier models from a motivational perspective, we explain the core attention mechanisms at the heart of the Transformer architecture. Then, we evaluate the performance of Transformers on a character-level, low-scale benchmark and compare them with RNNs. Finally, we discuss advantages and disadvantages of Transformers and give an overview of improvements since the original work [1] for future research in the field.

II. History

A. N-grams Models

n-grams models are one of the earliest techniques for language modeling, popularized by Shannon [2]. They are based on statistical assumptions such as the (first-order) Markov property [3]. Assume we have a sequence of length k, and we want to predict the probability distribution of the next, $(k+1)$-th token over the vocabulary. n-gram models approximate this distribution by conditioning only on the last n tokens (the Markov assumption).

An n-gram model defines a probability distribution over the next token given the previous $n-1$ tokens. That is, for every possible context sequence $v_1, \ldots, v_{n-1}$ of length $n-1$, it assigns a probability $P(v_n \mid v_1, \ldots, v_{n-1})$ to each token in the vocabulary conditioned on that context. Formally, for a token sequence $w_1, \ldots, w_{t-1}$ it approximates

$$P(w_t \mid w_1, \ldots, w_{t-1}) \approx P(w_t \mid w_{t-n+1}, \ldots, w_{t-1}). \quad (1)$$

From the conditional probabilities above, the probability of a token sequence of length l can be derived using the chain rule:

$$P(w_1, \ldots, w_l) = \prod_{i=1}^{l} P(w_i \mid w_{i-n+1}, \ldots, w_{i-1}) \quad (2)$$

The probability $P(v_n \mid v_1, \ldots, v_{n-1})$ is high if the token sequence $v_1, \ldots, v_n$ occurs frequently in the training text. Thus, this approach is largely based on counting and maintaining a large table of counts. Some implementations learn these statistics using gradient-based methods, but in general the model still relies on storing and maintaining such parameters after training.

The main limitations of n-gram models include:

- **Exponential growth:** Let V be the vocabulary. There are $|V|^{n-1}$ possible contexts, so the number of conditional distributions that must be stored grows exponentially with n.

- **Sparsity:** Most n-grams do not occur in the training corpus and therefore have a count of zero, which leads to many zero-probability estimates and inefficient use of memory.
- **Fixed context window:** The model conditions only on the previous $n-1$ tokens, so it cannot use information outside this window. For example, with trigrams ($n=3$), the next-token prediction for the sequence "No door is black" can only depend on "door is black" and cannot take the earlier word "No" into account, which semantically plays a huge role.
- **Poor generalization:** n-gram models treat different surface forms as unrelated. For example, "dog bites man" and "dogs bite a man" are treated as different sequences, and the model cannot naturally capture synonymy, morphology, paraphrases, or semantics.

B. RNNs

Recurrent neural networks (RNNs) are designed to be used with sequential data. Historically, they were motivated by early ideas in neural dynamics and associative memory, such as Hopfield networks [4], and by recurrent connectionist sequence machines such as Jordan networks [5]. Later, Elman popularized a simple recurrent architecture in which information from previous time steps is summarized in a hidden state [6].

Given an input sequence $x_1, \dots, x_T$, an RNN updates its hidden state h_t and produces an output y_t at each time step. A common (vanilla) RNN update is

$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h), \qquad y_t = W_{hy}h_t + b_y. \tag{3}$$

More generally, a recurrent layer defines a parametric state transition and readout,

$$h_t = f_\theta(x_t, h_{t-1}), \qquad y_t = g_\theta(h_t), \tag{4}$$

where f_θ is typically an elementwise nonlinearity applied to an affine transformation, and g_θ maps the hidden state to the desired output space. In practice, RNNs are often built by stacking multiple recurrent layers. For a L-layer RNN, the update at time step t for layer $\ell \in \{1, \dots, L\}$ can be written as

$$h_t^{(\ell)} = f_\theta^{(\ell)}\big(h_t^{(\ell-1)}, h_{t-1}^{(\ell)}\big), \qquad h_t^{(0)} = x_t, \tag{5}$$

and the network output is computed from the top layer, e.g., $y_t = g_\theta(h_t^{(L)})$. The earlier equation is the special case $L=1$ with $f_\theta(x_t, h_{t-1}) = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$.

RNNs are typically trained using backpropagation through time (BPTT). However, gradients can vanish or explode over long sequences, which makes learning long-range dependencies difficult [7].

The main limitations of vanilla RNNs include:

- **Sequential computation:** The hidden state must be computed step by step, which limits parallelization and increases training time.
- **Vanishing/exploding gradients:** When training with BPTT, gradients can decay or grow exponentially with sequence length, which makes optimization unstable and harms long-context learning.
- **Limited long-range memory:** In practice, information from far in the past is hard to preserve in h_t, so performance degrades when the task requires long-term context.

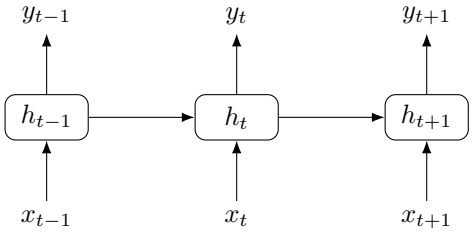

Fig. 1. An unrolled recurrent neural network over three time steps.

C. LSTMs

Long short-term memory (LSTM) networks were introduced by Hochreiter and Schmidhuber [8] to address the vanishing-gradient problem in vanilla RNNs and to enable learning over longer time spans. The key idea is to augment the recurrent hidden state with a memory cell c_t and a set of gates that regulate information flow.

Given an input x_t and the previous hidden state h_{t-1}, an LSTM computes the input gate i_t, forget gate f_t, output gate o_t, and a candidate cell update $\tilde{c}_t$ as

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \quad f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \tag{6}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o), \quad \tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c), \tag{7}$$

where $\sigma(\cdot)$ denotes the logistic sigmoid function. The cell state and hidden state are then updated via

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \qquad h_t = o_t \odot \tanh(c_t). \tag{8}$$

These gating mechanisms help preserve and selectively update information in c_t, which mitigates vanishing gradients and improves the ability to model long-range dependencies.

The main limitations of LSTMs include:

- **Sequential computation:** Like RNNs, LSTMs process sequences step by step, which limits parallelization.
- **Higher computational cost:** The gating operations increase the number of parameters and matrix multiplications compared to vanilla RNNs.
- **Long-context limitations remain:** While LSTMs substantially improve memory, performance can still degrade on very long sequences, and training can become slow for large corpora.

D. Limitations

- **n-gram models:** They use a fixed context window of $n-1$ tokens, which limits the ability to model long-range dependencies. They also suffer from sparsity as the required number of stored parameters grows exponentially with n.

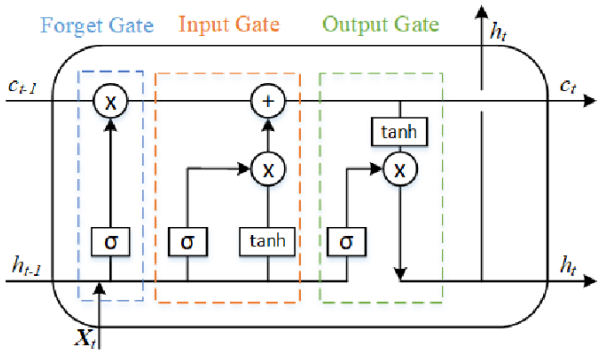

Fig. 2. LSTM architecture (adapted from [8]).

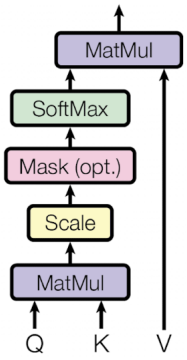

Fig. 3. Scaled dot-product attention (adapted from [1]).

- **RNNs:** They process sequences sequentially, which limits parallelization. Training with BPTT can lead to vanishing or exploding gradients, making it difficult to learn long-range dependencies.
- **LSTMs:** Gating improves long-range memory compared to vanilla RNNs, but computation remains sequential and more expensive due to additional parameters and operations. Very long contexts can still be challenging in practice.

E. Techniques to Address These Limitations

Techniques to mitigate the limitations above include:

- **n-gram models:** Laplace smoothing; backoff; interpolation; pruning; vocabulary restriction.
- **RNNs:** gradient clipping, truncated BPTT, careful initialization, normalization.
- **LSTMs:** GRUs, attention mechanisms, Transformer architectures.

III. Attention Mechanisms

A. General Formulation of Attention

In general, attention mechanisms are used to enrich the representation of sequence elements by aggregating information from another sequence (or the same sequence). Let the feature dimension be d and the sequence length be T, and let the inputs be $X_1, X_2 \in \mathbb{R}^{T\times d}$. A standard formulation projects these inputs into *queries*, *keys*, and *values* using learned projection matrices [1], [9], [10].

Historically, the idea of a content-based, differentiable read operation closely relates to early neural attention in encoder–decoder sequence-to-sequence models [9], [10] and to differentiable memory mechanisms in neural architectures such as Neural Turing Machines [11]. In these models, attention can be interpreted as a learned addressing scheme that retrieves and combines information from a set of vectors.

Let d_k denote the key/query dimension and let d_v denote the value dimension. We define

$$K = X_1 W_{\text{key}}, \qquad Q = X_2 W_{\text{query}}, \qquad V = X_1 W_{\text{value}}, \tag{9}$$

with projection matrices

$$W_{\text{key}} \in \mathbb{R}^{d\times d_k}, \qquad W_{\text{query}} \in \mathbb{R}^{d\times d_k}, \qquad W_{\text{value}} \in \mathbb{R}^{d\times d_v}, \tag{10}$$

so that $K, Q \in \mathbb{R}^{T\times d_k}$ and $V \in \mathbb{R}^{T\times d_v}$.

The intuition behind K is the question "What do I have?" that each element of X_1 asks about itself, while the intuition behind Q is the question "What am I looking for?" that each element of X_2 asks. The values V represent the information that will ultimately be aggregated.

To compute how strongly each query attends to each key, we form similarity scores (here: dot products) and normalize them with a row-wise softmax:

$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \in \mathbb{R}^{T\times T}. \tag{11}$$

The factor $\frac{1}{\sqrt{d_k}}$ is a crucial scaling term: for large d_k, dot products $q \cdot k$ tend to grow in magnitude, which can push the softmax into saturated regimes with very small gradients. Scaling the logits keeps their variance more stable and improves optimization, especially when using dot-product attention with high-dimensional key/query vectors. Finally, attention produces an enriched representation as a *weighted average* (weighted sum) of the value vectors:

$$Y = AV \in \mathbb{R}^{T\times d_v}. \tag{12}$$

Each output row y_t is a weighted sum of the rows of V, where the weights are given by the corresponding row of A. Since Q is computed from X_2, the output Y can be interpreted as an *enriched version of* X_2 using information retrieved from X_1.

In practice, Y is typically projected back to the original model dimension using an output projection $W_O \in \mathbb{R}^{d_v\times d}$:

$$\tilde{Y} = YW_O \in \mathbb{R}^{T\times d}. \tag{13}$$

B. Self-Attention

Self-attention is the special case of the general formulation where the sequence attends to itself, i.e., $X_1 = X_2 = X \in \mathbb{R}^{T\times d}$ [1]. In this case, each position in the sequence produces

a query and compares it to keys produced from the same sequence:

$$K = XW_{\text{key}}, \qquad Q = XW_{\text{query}}, \qquad V = XW_{\text{value}}. \quad (14)$$

The attention weights $A = \text{softmax}\big(\frac{QK^T}{\sqrt{d_k}}\big)$ are therefore computed entirely within the sequence, and the output

$$Y = AV \in \mathbb{R}^{T\times d_v} \quad (15)$$

is an enriched representation of the same sequence X. Intuitively, self-attention allows each token representation to incorporate information from other tokens that are most relevant for the current position. In an encoder-only Transformer, self-attention is typically applied to the input sequence to build contextual representations. In the original encoder–decoder Transformer [1], self-attention appears in both the encoder and the decoder. In the decoder, self-attention is usually *masked* to prevent attending to future positions during autoregressive generation.

C. Masked Attention

Masked (causal) attention is a variant of self-attention used in autoregressive decoders [1]. The key difference is whether a token at position t is allowed to attend to *future* positions $j > t$.

a) Bidirectional (unmasked) self-attention.: This case is identical to the self-attention formulation above. Here, each query position can attend to all key positions $1, \dots, T$, which yields *bidirectional* context (left and right).

b) Causal (masked) self-attention.: To enforce autoregressive generation, we apply an upper-triangular mask $M \in \mathbb{R}^{T\times T}$ that blocks attention to future tokens:

$$M_{tj} = \begin{cases} 0, & j \le t, \\ -\infty, & j > t. \end{cases} \quad (16)$$

The masked attention weights are then computed as

$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M\right) \in \mathbb{R}^{T\times T}, \quad (17)$$

so that $A_{tj} = 0$ for all $j > t$. The output shape is unchanged:

$$Y = AV \in \mathbb{R}^{T\times d_v}. \quad (18)$$

In practice, this masking ensures that the representation at position t depends only on tokens up to t, which matches the next-token prediction objective.

D. Cross-Attention

Cross-attention is the same as the general attention formulation, but with two different sequences, i.e., $X_1 \neq X_2$ in general. In this setting, X_2 provides queries, while X_1 provides keys and values, so the output is an enriched version of X_2 using information retrieved from X_1. In practice, cross-attention is used in encoder–decoder architectures, where X_1 typically comes from the encoder (source sequence) and X_2 from the decoder (target-side states) [1].

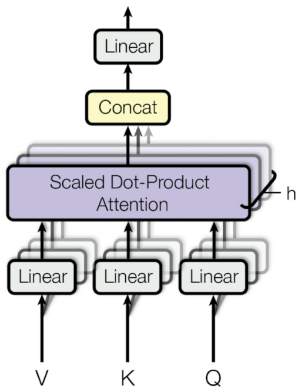

Fig. 4. Multi-head attention (adapted from [1]).

E. Multi-Head Attention

Multi-head attention applies the attention operation n times in parallel using different projection matrices [1]. Each head can focus on different relationships in the sequence, while keeping the computation fully differentiable.

a) Per-head projections.: Let the number of heads be n and let each head use the same dimensions d_k and d_v. For head $i \in \{1, \dots, n\}$, we use separate projections

$$W^{(i)}_{\text{key}} \in \mathbb{R}^{d\times d_k}, \qquad W^{(i)}_{\text{query}} \in \mathbb{R}^{d\times d_k}, \qquad W^{(i)}_{\text{value}} \in \mathbb{R}^{d\times d_v}. \quad (19)$$

Using the general attention notation with inputs $X_1, X_2 \in \mathbb{R}^{T\times d}$, head i computes

$$K^{(i)} = X_1 W^{(i)}_{\text{key}}, \qquad Q^{(i)} = X_2 W^{(i)}_{\text{query}}, \qquad V^{(i)} = X_1 W^{(i)}_{\text{value}}, \quad (20)$$

so that $K^{(i)}, Q^{(i)} \in \mathbb{R}^{T\times d_k}$ and $V^{(i)} \in \mathbb{R}^{T\times d_v}$.

b) Parallel attention heads.: Each head produces an output

$$Y^{(i)} = \text{softmax}\left(\frac{Q^{(i)}(K^{(i)})^T}{\sqrt{d_k}}\right) V^{(i)} \in \mathbb{R}^{T\times d_v}. \quad (21)$$

c) Concatenation and output projection.: The per-head outputs are concatenated along the feature dimension,

$$Y_{\text{cat}} = \text{Concat}\big(Y^{(1)}, \dots, Y^{(n)}\big) \in \mathbb{R}^{T\times (nd_v)}, \quad (22)$$

and projected back to the model dimension d using an output projection

$$\tilde{Y} = Y_{\text{cat}} W_O \in \mathbb{R}^{T\times d}, \qquad W_O \in \mathbb{R}^{(nd_v)\times d}. \quad (23)$$

d) Typical dimension choice.: Often $d_k = d_v = d/n$, so that $nd_v = d$ and the concatenation preserves the original feature dimension before the final projection.

IV. Transformer Architecture

A. Overall Model Structure

The Transformer is an encoder–decoder architecture built around attention mechanisms, originally proposed in [1]. Instead of recurrence, it uses stacks of attention and feed-forward sublayers to transform an input sequence into a sequence

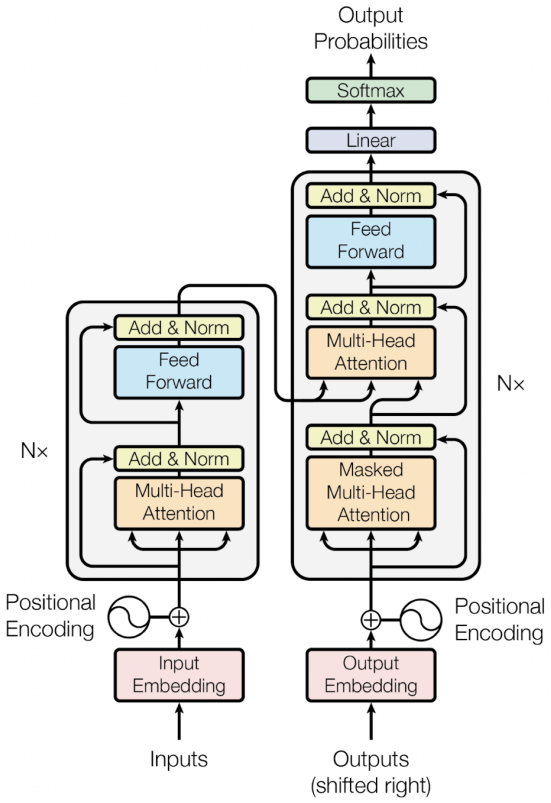

Fig. 5. Overall Transformer encoder–decoder architecture (adapted from [1]).

of contextual representations, and (optionally) to decode an output sequence autoregressively.

In the original Transformer, the from the authors suggested model dimension is $d = 512$ and the number of heads is $n = 8$, so that $d_k = d_v = d/n = 64$ [1].

a) Encoder–decoder framework.: Let $X \in \mathbb{R}^{T_{\text{src}} \times d}$ be the source sequence embeddings and let $H \in \mathbb{R}^{T_{\text{src}} \times d}$ denote the encoder output (often called "memory"). The decoder consumes a target sequence (shifted right during training) and produces decoder states $S \in \mathbb{R}^{T_{\text{tgt}} \times d}$, which are mapped to vocabulary logits.

b) Layer stacking.: Both encoder and decoder consist of L layers. Each encoder layer contains (i) multi-head self-attention and (ii) a position-wise feed-forward network (FFN). Each decoder layer contains (i) masked multi-head self-attention, (ii) encoder–decoder (cross) attention over H, and (iii) a position-wise FFN [1].

c) Residual connections and layer normalization.: Each sublayer is wrapped with a residual connection and layer normalization, and dropout is applied to sublayer outputs during training [1]. In compact form, for an input representation $U \in \mathbb{R}^{T \times d}$ and a sublayer function $\text{Sublayer}(\cdot)$ (attention or FFN), a typical *post-LayerNorm* (Add&Norm) update is written as

$$\text{LayerNorm}\big(U + \text{Sublayer}(U)\big), \tag{24}$$

which helps stabilize optimization and supports deep stacking.

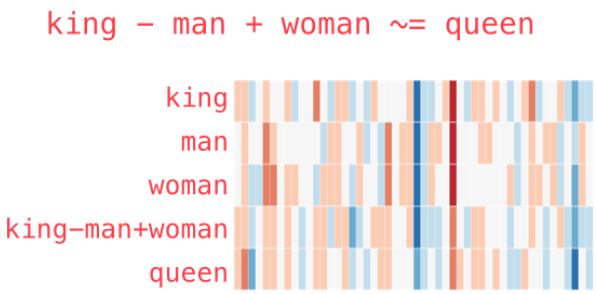

Fig. 6. Visualisation of word embeddings ([12]).

d) Post-LayerNorm vs. pre-LayerNorm.: The original Transformer in [1] uses post-LayerNorm, where normalization is applied after the residual addition. Many later implementations use *pre-LayerNorm*, where normalization is applied before the sublayer:

$$U + \text{Sublayer}(\text{LayerNorm}(U)). \tag{25}$$

Pre-LayerNorm helps the optimization to be more stable in deep Transformers, while post-LayerNorm matches the original definition from [1].

B. Positional Encodings

The Transformer processes all tokens in parallel and has no inherent notion of order. To be aware of position information, a positional encoding $p_t \in \mathbb{R}^d$ is added to each token embedding x_t, yielding $z_t = x_t + p_t$.

a) Fixed (sinusoidal) encodings.: The original Transformer [1] defines p_t using sine and cosine functions with geometrically increasing wavelengths. This construction introduces no trainable parameters and can extrapolate to sequence lengths not seen during training, since p_t can be computed for arbitrary t. Fig. 7 visualizes the resulting encoding matrix, where each row corresponds to a position and each column to a dimension; the alternating sine and cosine patterns at different frequencies are clearly visible.

b) Learned encodings.: Alternatively, p_t can be treated as a trainable parameter vector for each position up to a maximum length. Learned encodings can adapt to the statistics of a specific dataset and are used in many modern architectures, though they cannot generalize beyond the maximum position seen during training.

C. Encoder and Decoder Components

a) Encoder self-attention.: In each encoder layer, self-attention lets every source token attend to all other source tokens. This produces contextualized representations that can capture long-range dependencies in a single layer.

b) Decoder masked self-attention.: In each decoder layer, self-attention is *causal* (masked) so that position t may only use information from positions $\leq t$. This prevents the decoder from accessing future target tokens during training and generation.

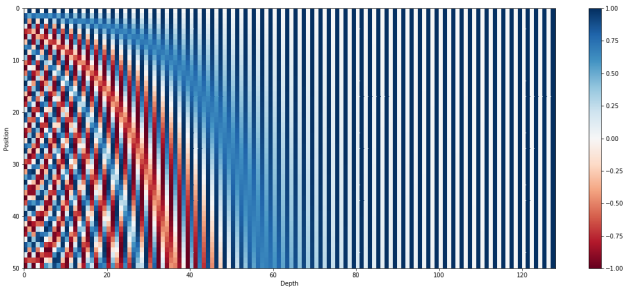

Fig. 7. The 128-dimensional positional encoding for a sentence with the maximum length of 50. Each row represents the embedding vector p_t (source: [13]).

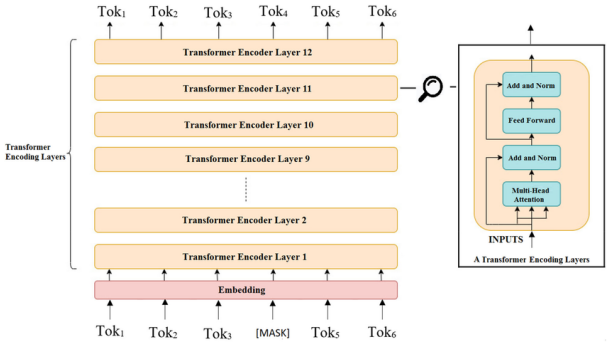

Fig. 8. BERT encoder-only Transformer architecture (source: [15]).

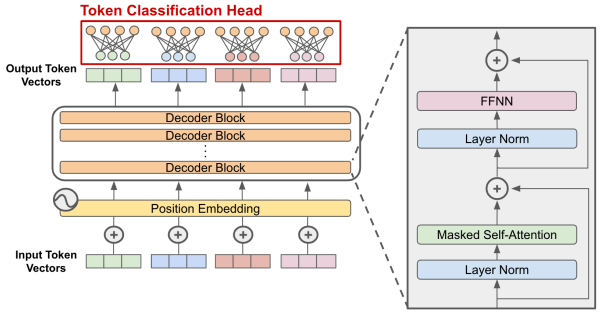

Fig. 9. Decoder-only Transformer architecture (adapted from [16]).

c) Encoder–decoder (cross) attention.: Cross-attention connects the decoder to the encoder output: each target position attends over the encoder "memory" to retrieve the most relevant source-side information, enabling conditioning on the full input (e.g., in machine translation).

D. Feed-Forward Networks

In addition to attention, each Transformer layer contains a position-wise feed-forward network (FFN) [1]. It is applied independently to each time step and therefore does not mix information across positions; instead, the FFN increases the representational capacity by transforming the feature vector at each token.

a) Position-wise MLP.: For an input representation $U \in \mathbb{R}^{T\times d}$, the FFN can be viewed as a two-layer MLP applied parallelly to every row of U:

$$\mathrm{FFN}(U) = \phi\big(UW_1 + b_1\big)W_2 + b_2. \tag{26}$$

Here, $W_1 \in \mathbb{R}^{d\times d_{\mathrm{ff}}}$ and $W_2 \in \mathbb{R}^{d_{\mathrm{ff}}\times d}$ are weight matrices, $b_1 \in \mathbb{R}^{d_{\mathrm{ff}}}$ and $b_2 \in \mathbb{R}^d$ are biases, and $\phi(\cdot)$ is an elementwise nonlinearity (ReLU in the original Transformer [1]).

b) Typical dimensions.: In the baseline setting of [1], the model dimension is $d = 512$ and the FFN hidden dimension is $d_{\mathrm{ff}} = 2048$, i.e., a $4\times$ expansion followed by a projection back to d. This expansion-contraction pattern is one reason why the FFN contributes a large fraction of parameters and compute in standard Transformer blocks.

c) Residuals and normalization.: As with attention, the FFN is wrapped in an Add&Norm block. Intuitively, the residual path preserves the current representation while the FFN provides a learnable, nonlinear "update" at each position

E. Encoder-only Transformer

Encoder-only Transformers retain only the encoder stack of the original architecture [1] and use *bidirectional* self-attention. The encoder consists of L identical layers, each containing multi-head self-attention and a position-wise FFN with residual connections and layer normalization. The output $H \in \mathbb{R}^{T\times d}$ provides contextualized representations that can be pooled or used per-token.

These models are typically trained with masked language modeling (MLM), where a fraction of tokens is masked and predicted from the full surrounding context [14]. They are well-suited for discriminative tasks such as text classification and sequence labeling, but not for autoregressive generation.

F. Decoder-only Transformer

Decoder-only Transformers retain only the decoder stack [1], using *causal* (masked) self-attention. Each of the L blocks contains masked multi-head self-attention and a position-wise FFN, wrapped with residual connections and layer normalization. After the final block, a linear head maps each position to vocabulary logits of dimension V.

The model is trained with next-token prediction using cross-entropy loss. At inference time, tokens are generated autoregressively by sampling from the logits at the last position. This architecture is the basis of modern LLMs such as GPT-style models [16].

V. Advantages and Disadvantages of Transformers

A. Advantages

a) Parallelism.: Unlike RNNs and LSTMs, Transformers process all positions in a sequence simultaneously, which allows full utilization of modern GPU and TPU hardware and leads to substantially faster training.

b) Scalability.: Empirical scaling laws show that Transformer performance improves predictably with model size, dataset size, and compute [17], which has enabled the training of models with hundreds of billions of parameters.

TABLE I
PER-LAYER COMPLEXITY COMPARISON, WHERE T IS THE SEQUENCE LENGTH AND d IS THE REPRESENTATION DIMENSION (ADAPTED FROM [1]).

Layer Type	Complexity per Layer	Sequential Ops	Max Path Length
Self-Attention	$O(T^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(T \cdot d^2)$	$O(T)$	$O(T)$

c) Flexibility.: The same core architecture can be used for language modeling, machine translation, classification, vision, and multimodal tasks by changing only the input representation and training objective.

B. Disadvantages

a) Quadratic complexity.: Self-attention computes pairwise scores for all T^2 token pairs, which makes both memory and compute grow quadratically with sequence length and limits the practical context window.

b) Data and compute requirements.: Transformers typically require large-scale datasets and significant computational resources to reach competitive performance, raising concerns about energy consumption and accessibility.

c) Interpretability.: Although attention weights are sometimes used as explanations, they do not always correspond to the features that drive predictions, and the overall decision process of deep Transformer stacks remains difficult to interpret.

VI. EMPIRICAL EVALUATION

A. Evaluation Metrics

Several standard metrics are used to evaluate language models and sequence-to-sequence systems:

- **Perplexity** measures how good a language model can predict a test text. It is simply the exponent of average negative log-likelihood per token.
- **BLEU** is a precision-based metric for machine translation tasks [1].
- **Accuracy and F1** are used for classification and sequence-labeling tasks. Accuracy measures the fraction of correct predictions, while F1 balances precision and recall, which is especially useful for imbalanced label distributions.

B. Comparison with Alternative Models

RNNs and Transformer-based models approach next-token prediction differently. An RNN compresses the entire history into a fixed-size hidden state h_t, so each prediction is conditioned on a lossy summary of the past. A decoder-only Transformer, by contrast, attends directly to all previous tokens via causal self-attention, giving it access to richer context at the cost of quadratic complexity in the sequence length.

On large-scale benchmarks the Transformer consistently outperforms recurrent models [1]. However, on small datasets and with small model budgets the picture is more nuanced. Transformers have more parameters per layer and could overfit or underfit when data is sparse, whereas a compact RNN can learn reasonable sequential statistics with fewer parameters. To empirically show this, we trained both architectures on the same character-level Shakespeare corpus for five epochs using different hyperparameter configurations. Table II summarizes the results for the selected hyperparameter configurations.

TABLE II
CHARACTER-LEVEL NEXT-TOKEN PREDICTION ON THE SHAKESPEARE DATASET (5 EPOCHS). ALL MODELS USE A TEST/TRAIN RATIO OF 0.8.

Model	Layers	Emb	Seq	Batch	Params	Time	Train Loss	Test Loss	Test Acc
RNN[a]	2	64	128	128	70k	30m	1.32	1.32	0.592
RNN[a]	2	64	512	512	70k	2h 6m	1.30	1.35	0.580
Transformer[b]	2	64	128	128	84k	17m	1.76	1.60	0.513
Transformer[c]	2	64	512	512	108k	1h 37m	1.88	1.70	0.489

[a]Hidden size 128. [b]FFN dim 128, 8 heads. [c]FFN dim 128, 2 heads.

VII. APPLICATIONS

A. Natural Language Processing

Transformers have become the dominant architecture across NLP. Encoder-only models such as BERT [14] are widely used for text classification, named-entity recognition, and question answering, where bidirectional context is essential. Decoder-only models power large-scale language generation, dialogue systems, and code completion. Full encoder–decoder Transformers remain the standard for sequence-to-sequence tasks such as machine translation and abstractive summarization [1].

B. Computer Vision

The Vision Transformer (ViT) splits an image into fixed-size patches, embeds each patch as a token, and processes the resulting sequence with a standard Transformer encoder. This approach matches or surpasses convolutional networks on image classification when trained on sufficiently large datasets. Variants have since been applied to object detection, semantic segmentation, and video understanding.

C. Other Domains

Beyond language and vision, Transformers have also been adopted in speech recognition, where they mostly replace recurrent encoders, in protein structure prediction, where attention over amino-acid sequences captures long-range biochemical interactions, and in time-series models where the attention is able to capture dependencies across timestamps.

VIII. LATER IMPROVEMENTS

A. Efficient Attention Variants

The quadratic cost of self-attention has motivated a range of approximations. Sparse attention patterns, as in the Reformer [18], restrict each token to attend to a subset of positions, reducing complexity toward $O(T \log T)$ or $O(T\sqrt{T})$. Reversible layers allow activations to be recomputed during the backward pass instead of stored, which reduces memory consumption from $O(L)$ to $O(1)$ in the number of layers. Chunked feed-forward layers split the sequence into smaller chunks and

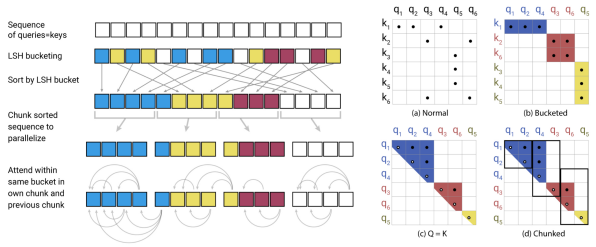

Fig. 10. Locality-sensitive hashing (LSH) attention as used in the Reformer [18].

TABLE III
CONFIGURATIONS OF THE MAIN BERT AND ALBERT MODELS ANALYZED IN [19]

Model	Size	Parameters	Layers	Hidden	Embedding	Parameter-sharing
BERT	base	108M	12	768	768	False
	large	334M	24	1024	1024	False
ALBERT	base	12M	12	768	128	True
	large	18M	24	1024	128	True
	xlarge	60M	24	2048	128	True
	xxlarge	235M	12	4096	128	True

process each chunk independently, trading an increase in computation for a significant reduction in peak memory usage.

B. Architectural Modifications

Several directions aim to make Transformers smaller or more efficient without sacrificing quality. Parameter sharing across layers, as in ALBERT [19], drastically reduces model size while retaining most of the performance. Factorized embedding parameterization, also introduced in ALBERT [19], maps the tokens from the vocabulary embedding dimension to the hidden dimension using a intermediate hidden dimension, which significantly reduces the number of parameters in the embedding layer. On training objective level, alternatives to the original masked language modeling (MLM) have been explored: BERT's next sentence prediction (NSP) was replaced by the sentence order prediction (SOP) task in ALBERT, which proved more effective at capturing inter-sentence coherence.

IX. CONCLUSION

In this paper, we gave an overview of attention mechanisms and the Transformer architecture. We mentioned the development of language modeling approaches from n-gram models through RNNs and LSTMs to the self-attention and Transformers, and described the key components—scaled dot-product attention, multi-head attention, positional encodings, and feed-forward networks, that make up encoder-only, decoder-only, and full encoder-decoder models.

Our empirical comparison on a small-scale character-level task showed that recurrent models can remain competitive with Transformers when data and model budgets are limited, while Transformers excel as both scale. The advantages of Transformers—full parallelism, direct long-range token interactions, and predictable scaling behavior—come at the cost of quadratic complexity in sequence length and high data and compute requirements.

Efficient attention variants, parameter-sharing strategies, and hybrid architectures promise to address these limitations. Since Transformers continue to expand into domains beyond NLP, including vision, speech a solid understanding of their core mechanisms remains essential.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[2] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, 1948.

[3] A. A. Markov, "An example of statistical investigation of the text eugene onegin concerning the connection of samples in chains," 1913.

[4] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[5] M. I. Jordan, "Attractor dynamics and parallelism in a connectionist sequential machine," in *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, 1986.

[6] J. L. Elman, "Finding structure in time," in *Cognitive Science*, vol. 14, no. 2, 1990, pp. 179–211.

[7] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," in *IEEE Transactions on Neural Networks*, vol. 5, no. 2, 1994, pp. 157–166.

[8] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[9] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[10] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *International Conference on Learning Representations*, 2015.

[11] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," in *arXiv preprint arXiv:1410.5401*, 2014.

[12] "Visual embeddings king-queen example," ResearchGate figure. [Online]. Available: https://jalammar.github.io/illustrated-word2vec/$0

[13] A. Kazemnejad, "Transformer architecture: The positional encoding," Blog post, 2019.

[14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *Proceedings of NAACL-HLT*, pp. 4171–4186, 2019.

[15] "Bert model architecture," ResearchGate figure, 2019. [Online]. Available: https://www.researchgate.net/figure/BERT-model-architecture_fig2_372906672

[16] C. R. Wolfe, "Decoder-only transformers: The workhorse of generative LLMs," Substack: Deep (Learning) Focus, 2024. [Online]. Available: https://cameronrwolfe.substack.com/p/decoder-only-transformers-the-workhorse

[17] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown *et al.*, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[18] N. Kitaev, Ł. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," *International Conference on Learning Representations*, 2020.

[19] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *International Conference on Learning Representations*, 2020.