TRANSFORMERS UNITED

Thomas Dooms, Alexander Belooussov

University of Antwerp

ABSTRACT

The field of natural language processing (NLP) has seen remarkable progress with the invention of attention mechanisms and transformers. These technologies caused researchers in other areas of machine learning to also explore these models to improve their own results. Attention mechanisms enable models to dynamically focus on specific parts of an input. While, on the other hand, transformers are a modern neural network architecture that uses these attention mechanisms combined with novel normalization techniques to process sequential data in an optimized manner. In this work, we will take a closer look at these technologies, investigating architectural variations and applications. We will also delve into the recent advancements in the field of image synthesis, where machine-learning models are used to generate new images from a provided dataset. We will examine the technical capabilities and limitations of this technology, as well as its potential applications. Overall, we provide a comprehensive overview of attention mechanisms, transformers, diverse use cases, and image synthesis, highlighting their current state-of-the-art and future directions.

Index Terms— Attention, Transformers, Vision, Reinforcement Learning, Diffusion Models

1. INTRODUCTION

The use of recurrent and convolutional architectures was once state-of-the-art in various fields of machine learning. However, the introduction of the "Attention is All You Need" paper [1] marked a significant shift in the field. While the paper primarily focused on natural language processing and machine translation, it severely impacted other areas of machine learning as well. The attention mechanism, which had already been introduced [2], was utilized in a novel way in the "Attention is All You Need" paper. The authors used it to replace the complex recurrent architectures in the encoder and decoder of their machine translation architecture with multi-headed self-attention (see section 2). This innovation allowed for faster training and resulted in a new state-of-the-art in machine translation.

Since the release of the "Attention is All You Need" paper, transformers have continued to evolve and have been applied in a variety of fields. Some notable examples include the GPT architectures for text generation and diffusion models for image generation. Both have gained significant attention and popularity on social media, with models such as ChatGPT (a research preview and sibling of InstructGPT [3]) and DALL-E 2 [4] receiving widespread attention. In this work, we will discuss image generation in more detail and also provide an overview of key research in the evolution of transformers, including their use in fields such as natural language processing, text generation, image classification, and reinforcement learning. It is worth noting that transformers have been adapted in numerous works and our analysis only covers a small portion of this research. For example, Figure 1 [5] illustrates the extent of research that has been conducted on improving the efficiency of the transformer architecture.

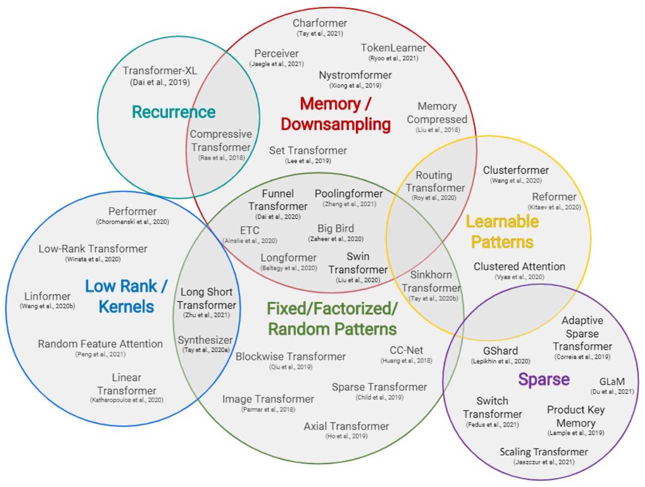
Charformer (Tay et al., 2021)
Perceiver (Jaegle et al., 2021)
TokenLearner (Ryoo et al., 2021)
Transformer-XL (Dai et al., 2019)
Nystromformer (Xiong et al., 2019)
Recurrence
Memory / Downsampling
Memory Compressed (Liu et al., 2018)
Compressive Transformer (Rae et al., 2018)
Set Transformer (Lee et al., 2019)
Routing Transformer (Roy et al., 2020)
Clusterformer (Wang et al., 2020)
Reformer (Kitaev et al., 2020)
Funnel Transformer (Dai et al., 2020)
Poolingformer (Zhang et al., 2021)
Performer (Choromanski et al., 2020)
ETC (Ainslie et al., 2020)
Big Bird (Zaheer et al., 2020)
Learnable Patterns
Low-Rank Transformer (Winata et al., 2020)
Longformer (Beltagy et al., 2020)
Swin Transformer (Liu et al., 2020)
Clustered Attention (Vyas et al., 2020)
Sinkhorn Transformer (Tay et al., 2020b)
Linformer (Wang et al., 2020b)
Low Rank / Kernels
Long Short Transformer (Zhu et al., 2021)
Fixed/Factorized/ Random Patterns
Adaptive Sparse Transformer (Correia et al., 2019)
Random Feature Attention (Peng et al., 2021)
Synthesizer (Tay et al., 2020a)
Blockwise Transformer (Qiu et al., 2019)
CC-Net (Huang et al., 2018)
GShard (Lepikhin et al., 2020)
Sparse
GLaM (Du et al., 2021)
Linear Transformer (Katharopoulos et al., 2020)
Sparse Transformer (Child et al., 2019)
Image Transformer (Parmar et al., 2018)
Switch Transformer (Fedus et al., 2021)
Product Key Memory (Lample et al., 2019)
Axial Transformer (Ho et al., 2019)
Scaling Transformer (Jaszczur et al., 2021)

Fig. 1. Different efficient transformer networks from [5]

While there are many topics related to transformers that could be studied, we have chosen to focus on the field of image generation. This decision is partly due to the widespread popularity of models such as DALL-E 2 [4], Midjourney [6], and Imagen [7]. Prior to the emergence of transformers and diffusion models, generative adversarial networks (GANs) [8] were considered state-of-the-art, with notable examples including BigGAN [9] and StyleGAN [10]. GANs have been widely studied by the research community and have achieved satisfactory scores on various image-generation metrics. They also have the advantage of being efficient at sample time. However, GANs have some notable limitations. One issue is that their training procedure is highly dependent on the proper initialization of hyperparameters. Additionally, while GANs can generate high-quality samples, they often lack diversity, resulting in generated samples that are correct but not very distinct from each other. Diffusion models aim to address these issues, starting with the introduction of denoising diffusion probabilistic models [11], which incorporated attention into the U-net architecture [12] (more on this in section 6). In recent years, research in this field has accelerated significantly and has continued to make progress, even surpassing the performance of GANs in some cases [13]. This has included the incorporation of transformer blocks into the architecture [14] and the development of the widely recognized image generation models we are familiar with today.

In the following section, we will provide an easy-to-follow introduction to the concept of attention and several variations to the attention mechanism which are used in transformers and image synthesis models. In section 3, we will present a detailed description of the transformer architecture and its inner workings. In section 5, we will examine a variety of applications of transformers in different fields of machine learning. We will then perform a comprehensive analysis of the history and current state of diffusion models in section 6, including the role of transformers in image generation.

2. ATTENTION

2.1. Motivation

To fully understand the operation of transformers, it is necessary to grasp the fundamental concept of attention. Attention mechanisms allow models to assign varying levels of importance to different input elements, similar to how humans do. For example, when translating a sentence from Latin to English, some parts of the sentence may be more significant than others. While many words have a straightforward one-to-one correspondence between languages, the presence of a Latin verb can significantly alter the meaning of the entire English sentence. Therefore, when translating this text, it is crucial to pay particular attention to the current word being translated as well as the verb. This technique addresses the prominent issue of recurrent networks and similar architectures forgetting distant information by increasing the importance of relevant context from previous sentences or sections. The drawback of this method is that it carries a high computational cost, with the complexity of calculating these weights being quadratic rather than linear. This can make it prohibitively expensive for many tasks.

2.2. Keys, Queries and Values

In attention mechanisms, queries, keys, and values are used to calculate the attention weights that determine which parts of the input the model should focus on. Conceptually, queries are used to determine which parts of the input the model should attend to, while keys, similar to their use in data structures, are used to index the values. Queries and keys are typically a vector representation of a specific element in an input sequence, such as a word in a sentence. Finally, values are the actual input elements that the model is attending to. They can be thought of as the content that the model uses to update its internal state or generate output.

To calculate the attention weights, the model compares the similarity of the queries and the keys using a dot-product. These weights represent the relative importance of each input element and are used to weigh the values. Formally, the full attention function can be written as:

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

Here, Q is the query matrix, K is the key matrix, V is the value matrix where each row corresponds to an input token, and d_k is the dimensionality of the keys. The dot product of Q and K is divided by the square root of d_k which can be considered to be a kind of normalization on the dimensions, then, the softmax function is applied to the previous value to compute the final attention weights. These attention weights are then used to weigh the values in the matrix V component-wise, and the resulting weighted sum is the output of the attention mechanism [15, 1].

This sequence of operations, which computes the attention in each layer of the transformer, is known as a self-attention head. This is typically depicted as shown on the left side of Figure 2.

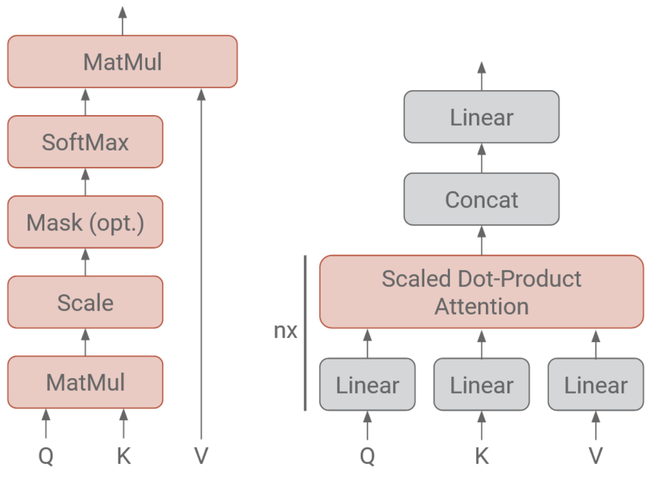

Fig. 2. Mutli-head attention

3. TRANSFORMERS

A diagram of the general architecture can be found in Figure 3. At first glance, the transformer architecture may appear complex due to the numerous components and connections. Some of these components, such as the self-attention block, will be clearer once we delve into the full architecture and its training procedure. It is recommended for those new to transformers to review this section twice.

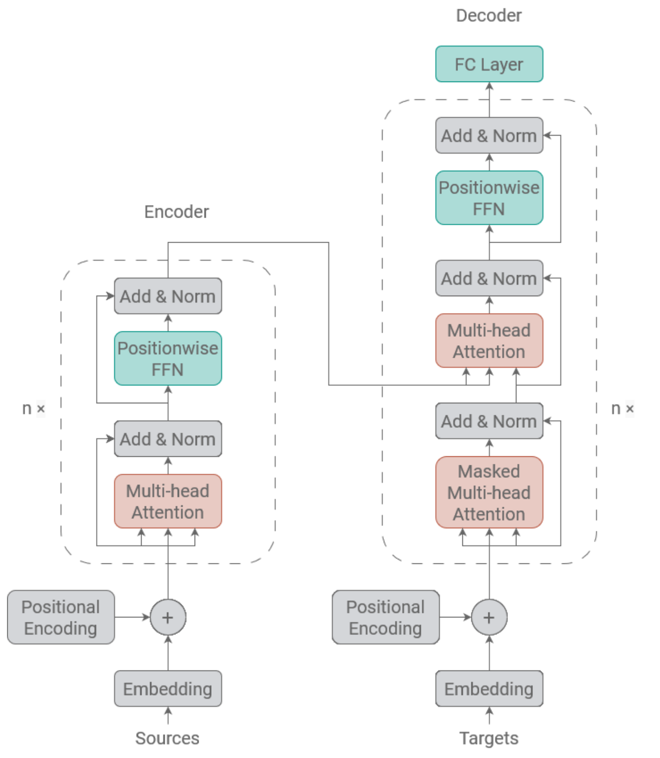

Fig. 3. The transformer architecture

3.1. Embeddings

Transformers cannot be trained on raw text. Instead, tokenizers are used to process the text into a format that can be used. In the case of NLP, tokenization typically consists of a few steps, including normalization, pre-tokenization, a pass through a model, and post-processing. Normalization cleans up the input, such as by removing accents and making everything lowercase. Pre-tokenization splits the input into segments such as words. The model is trained on the corpus to split the segments further (if necessary) into individual tokens and convert them into integer IDs. Post-processing adds any special tokens such as padding or separators.

The IDs are passed through a learned linear transformation such that each token is represented using a vector embedding of size d_{embed}. On its own, the transformer architecture does not consider the order of elements in a sequence because it does not use convolutions or recurrence. This is a problem since the order of words can significantly change the meaning of a sentence (e.g. the placement of the word "not"). This is solved by simply adding a positional encoding to the embedding. This encoding is a vector of the same length as the embedding. This can be done using d_{embed} sine and cosine functions of different frequencies [1]. The elements in the positional encoding are then given by the following formulas:

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

Here, pos is the index of the token in the sequence, and $2i$ and $2i+1$ are the dimensions of the embedding, for $i = 0, 0.5, ..., \frac{d_{embed}-1}{2}$. By simply adding the encoding to the embedding of the token, the transformer is able to learn the order within the sequence.

3.2. Multi-Head Attention

Transformers utilize a more complex form of attention called multi-head attention (see Figure 2). In multi-head attention, the attention mechanism is applied multiple times to the input in parallel. Conceptually, each attention head acts like a separate attention mechanism, with its own queries, keys, and values.

To provide a concrete example, consider the previous task of translating a sentence from Latin to English. In this case, multiple attention heads might focus on different aspects of the sentence, such as remembering singular or plural forms or differentiating between masculine and feminine nouns. More complex relationships might also be learned, such as the word "caput," which can refer to both the body part "head" and the "capital city" in Latin. In some cases, the necessary context for disambiguation may be located far away in the input and require a specialized context head to evaluate it. In summary, the use of multiple attention heads allows the model to learn more complex relationships between input elements and attend to multiple, potentially conflicting, aspects of the input at the same time.

To implement multi-head attention, the queries, keys, and values are transformed with distinct linear layers for each attention head. The attention mechanism is then applied to each transformed embedding. The outputs of the different attention heads are concatenated and combined using another linear transformation to produce the final output of the multi-head attention layer.

3.3. Cross-Attention

Another type of attention used in transformers is cross-attention, in which the keys and values are from a different source than the queries. This is primarily used in sequence-to-sequence tasks such as translation and summarization, where the keys and values are computed from the input and the queries are computed from the output generated thus far.

This enables the model to examine the values from the input embeddings for information that is most relevant to the query vector. Recently, this technique has been utilized in text-guided image synthesis, enabling the image-generation process to consider the specified prompt more carefully.

3.4. Position-Wise Feed-Forward Network

The position-wise feed-forward network (FFN) is a neural net consisting of two fully-connected layers. The hidden layer dimension is typically around 4 times larger than the embedding size. The output is once again the size of the embeddings. An activation function is applied to the activations of the hidden layer. This block can be summarised using the following equation:

$$FFN(x) = activation(xW_1 + b_1)W_2 + b_2$$

Where $W_1 \in \mathbb{R}^{d_{embed} \times d_f}$, $W_2 \in \mathbb{R}^{d_f \times d_{embed}}$, b_1, and b_2 are learnable parameters. The activation function used in the model can be any of the commonly employed functions such as Rectified Linear Unit (RELU) or Gaussian Error Linear Unit (GELU), with the latter being frequently utilized in most transformer-based models [16, 17]. This is given by the formula:

$$GELU(x) = x\Phi(x)$$

Where $\Phi(x) = P(X <= x), X \sim \mathcal{N}(0, 1)$ is the cumulative distribution function of the standard normal distribution [18]. Each encoder/decoder layer has its own FFN. It is called position-wise because it is applied to every (processed) token, at every position, equally without any influence from other tokens.

3.5. Layernorm

Normalization has long been used in neural networks to reduce variance and improve training times. Batch normalization, introduced in 2015 [19], is the most commonly used normalization layer. It normalizes the values in the batch and sentence length dimensions to reduce variance: Each feature of a token is normalized based on its occurrence in other parts of the input and batch, rather than being normalized in relation to other features of the same token. While this method has been shown to outperform previous techniques, it has a major drawback: at inference time, there are no batches, so a moving average must be used for normalization instead.

Transformers take a different approach by using layer normalization, which is applied in the feature and sentence length dimensions. This does not take into account any other sequences in the batch but rather normalizes based on all the contents of a single input sequence. If H is the length of the output of the layer, x_i is the ith output of the layer, and ϵ is a small constant used for smoothing near 0, the normalization is given by the following equations:

$$\mu = \frac{1}{H}\sum_{i=1}^{H} x_i$$

$$\sigma^2 = \frac{1}{H}\sum_{i=1}^{H}(x_i - \mu)^2$$

$$layernorm(x_i) = \frac{x_i - \mu}{\sigma + \epsilon}$$

The goal of layer normalization is to ensure that the inputs to each layer have a mean of zero and a unit variance, which helps stabilize the distribution of activations and can improve the performance of the model. In the context of transformers, layer normalization is applied to the output of the self-attention layers and the output of the feed-forward layers. Additionally, the input to a layer is added as a residual connection: $layernorm(x + sublayer(x))$ [1]. This helps ensure that the activations in the self-attention layers and the feed-forward layers have a stable distribution, which improves the performance of the model. In recent years, many architectures have adopted this form of normalization [20].

3.6. Encoder

The encoder receives the input embeddings (e.g. the text to be translated) summed with a positional encoding, as mentioned previously. An encoder layer consists of a multi-head attention layer, a layernorm layer (with residual connection), a position-wise feed-forward network, and a second layernorm layer (with residual connection). Transformer architectures stack a number of these encoder layers on top of each other. For sequence-to-sequence tasks such as translation, the output of the final encoder layer is used to generate the keys and values for the cross-attention layers in the decoder.

Some architectures consist of only encoder layers, these typically excel at contextual understanding tasks such as text classification (eg. sentiment analysis). See subsection 5.1 for an example of this architecture.

3.7. Decoder

The decoder block is similar to the encoder block, with a few key differences. The input to a decoder layer is the generated sequence thus far, and in the case of training, the sequence which should be generated. The primary difference is the use of masking within the initial multi-head attention layer during the training process. This measure is taken to prevent the attention layer from accessing “future” (or “target”) tokens when making predictions. The second difference is the utilization of a cross-attention layer, which enables the model to consider the input at each step of the generation process. As mentioned in the previous section, it generates the keys and values from the output of the encoder. Decoder layers are also stacked in the same way as encoder layers.

As with encoders, some architectures only use decoder layers. However, since there is no encoder output, the cross-attention layer is omitted. These types of architectures are powerful sequence generators. See subsection 5.1 for an example.

3.8. Training & Details

A major advantage of transformers compared to RNNs is that they can be trained more efficiently. When training a language model for translation, the original sentence is fed to the source, and the translated sentence is fed to the targets (see Figure 3). To train on the whole sentence, the decoder part is masked differently according to the position of the word so that it cannot look into the future. Because this process depends only on the current position, it can be parallelized while an ordinary RNN must be trained fully sequentially. Further optimizations are possible; the encoder part of a sentence doesn't change during training. This can be cached such that this doesn't need to be recalculated [1].

4. IMPROVING TRANSFORMERS

4.1. Scaling Transformers

4.1.1. Bottlenecks

In recent years, scaling neural networks has proven fruitful in the field of machine learning. Despite the success that has been achieved through this process, it has not been without its challenges. The ability to train larger networks is highly dependent on so-called 'bottlenecks'. For example, in the past, the inability to train deep networks was a significant hurdle. However, this problem has been partially addressed through the introduction of techniques such as residual connections and effective regularization techniques applied after each layer. These deeper networks result in CNN architectures having a greater perceptive field, which allows them to learn more long-distance relations [21].

Until recently, NLP in particular had a serious bottleneck: the inability to handle large context sizes. The recurrent nature of LSTMs and other prominent architectures, made the network forget words or events that happened a while back. Empirical studies have shown that the accuracy of these models plateaued after a context size of approximately 100 tokens, regardless of the model size [21]. This changed with the introduction of transformers [1] which had the ability to directly look at the past. This has enabled the utilization of larger datasets and models, leading to an increase in the accuracy of NLP tasks without an apparent upper limit.

Curating these ever-increasing datasets quickly became unattainable for humans. This led to a shift towards the use of web-scraped datasets, which often consist of billions of images and trillions of tokens. To process these large datasets, transformer architectures also grew in size, with an approximate increase of 5x per year. The training of these large models often requires the use of superclusters of GPUs and incurs significant electricity costs. As a result, many researchers in the field have begun to closely examine the learning process of transformers and explore ways to improve both compute and data efficiency.

4.1.2. Switch Transformer

One way to improve efficiency is by introducing sparsity in the network. This sparsity can be introduced in self-attention as will be covered in the next section or by partitioning the network into multiple components. This latter method is known as a Mixture of Experts (MoE). These components can be used in two ways: sequentially or in parallel. A sequential mixture of experts routes the input to a select group of experts which generates the output, this allows the number of parameters of the model to be scaled (by introducing more experts) without increasing compute. In contrast, a parallel mixture of experts uses all experts and combines their knowledge afterward, this method does not induce sparsity but can improve accuracy. One can notice that multi-head attention is a form of a parallel mixture of experts.

The authors of the Switch Transformer architecture [22], use a sequential Mixture of Expert architecture within each transformer layer as seen in Figure 4. Self-attention is applied as usual but, instead of having a single feedforward layer, multiple experts are used with a router that decides which path is taken. Using different experts has its challenges, namely increased communication overhead and high training instability. The former is solved by selectively decreasing floating point precision and the latter by careful initialization of the parameters.

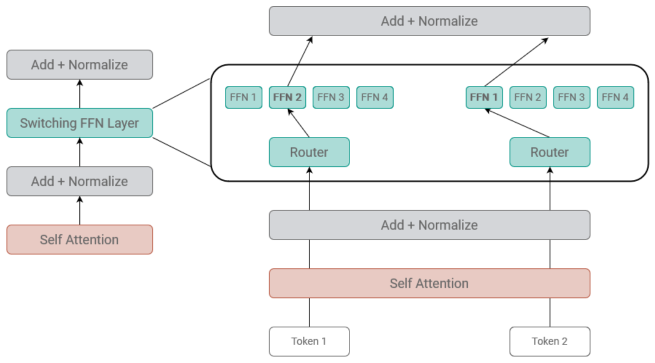

Fig. 4. The switch transformer

Model architecture	Complexity	Sequential Op.
RNN [23]	$O(n)$	$O(n)$
Transformer [1]	$O(n^2)$	$O(1)$
Reformer [24]	$O(nlog(n))$	$O(log(n))$
Linformer [25]	$O(n)$	$O(1)$
Performer [26]	$O(n)$	$O(1)$
Sparse Transformer [27]	$O(n\sqrt{n})$	$O(1)$
Big Bird [28]	$O(n)$	$O(1)$
Perceiver [29]	$O(n)$	$O(1)$

Table 1. Complexity per layer and sequential operation complexity for several variations on transformers

4.2. Avoiding Squared Self-Attention Complexity

4.2.1. Reducing Complexity

As previously discussed in various sections of the paper, self-attention has a quadratic complexity, which poses a significant limitation on the types of input that can be used in comparison to CNNs or other machine learning models. This has prompted researchers to seek out better architectures that combine the flexibility of transformers with the speed of CNNs. This section will give a comprehensive overview of different strategies to achieve this. Table 1 provides an overview of the different models that will be covered and their complexity.

The table is split into three sections. The first section contains the RNN and the transformer as a reference: the RNN has a sequential operation complexity of $O(n)$ as it must be performed successively on each token. In contrast, the transformer can operate on the whole input at once, with the caveat that the complexity of each layer is squared. The next section contains several architectures which try to reduce self-attention based on assumptions or approximations. The last section will cover other architectures which try to downsample in various ways or avoid the complexity altogether.

4.2.2. Reformer

The attention formula, ignoring multiple heads, is commonly noted as:

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

The expensive aspect of the attention formula, both in terms of memory and computation, is the multiplication of the queries and the keys, which results in an $N \times N$ matrix. Researchers at Google Research sought to remove this expensive multiplication by noting that the softmax around QK^T will reduce almost every entry in the matrix to negligibly small values except a few. Only entries with a high similarity between the key and the query will dominate. If the number of keys to be evaluated could be reduced beforehand, it would result in significant speed-ups. They decided to address this issue by using locality-sensitive hashing (LSH), which is able to map keys and queries into several buckets. LSH has two guarantees: it puts similar embeddings into the same bucket with high probability and dissimilar embeddings into distinct buckets with high probability. Once the buckets are determined, they are sorted and chunked, as shown in Figure 5. The use of chunks in LSH is a crucial aspect of the Reformer architecture, as it allows for a controlled reduction of the quadratic complexity of the self-attention mechanism. Without the restriction of chunking, LSH could result in a worst-case complexity of $O(n^2)$, which would defeat the purpose of using LSH to improve the efficiency of the transformer. It is important to note that when optimizing for loss, the transformer has a tendency to favor larger buckets as it gains access to the full, more powerful self-attention mechanism. This highlights the importance of carefully controlling the complexity of the model through techniques such as chunking to achieve the desired level of performance and efficiency. Self-attention is not performed within buckets but within chunks, as can be seen in Figure 5. Additionally, self-attention is also performed on tokens of the previous chunk if they originate from the same bucket; this is illustrated with both the green and orange buckets. There are additional details, such as what to do when a bucket contains only queries and no keys, or how this is backpropagated, which will not be covered here [24].

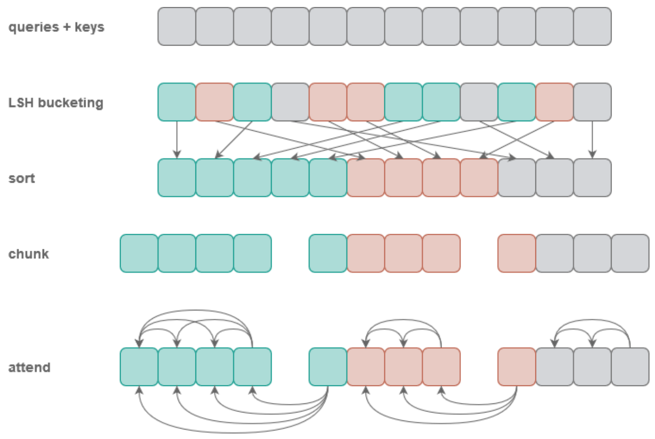

Fig. 5. Modified self-attention with LSH buckets

4.2.3. Linformer

The previously proposed architecture was found to possess several limitations. Despite its $O(nlog(n))$ complexity, it was only slightly faster than a traditional transformer on a large scale. Additionally, it would be desirable to achieve linear complexity, similar to that of CNNs. The Linformer paper aims to address these shortcomings. In the context of multi-head attention, it was determined that self-attention is approximately low-rank. This implies that the matrix is larger than the information it represents, which can be exploited to reduce dimensionality with negligible loss. This hypothesis was verified through a study of eigenvalues on the attention weights of the RoBERTa-Large model. Eigenvalues close to zero indicate an approximately lower rank matrix. The study found that the weights were only approximately rank 128, instead of 512. Using this understanding, the authors modify the attention mechanism to include a projection of the values and keys into a constant-sized latent space (128 in this case) as seen in Figure 6, thereby removing the quadratic complexity.

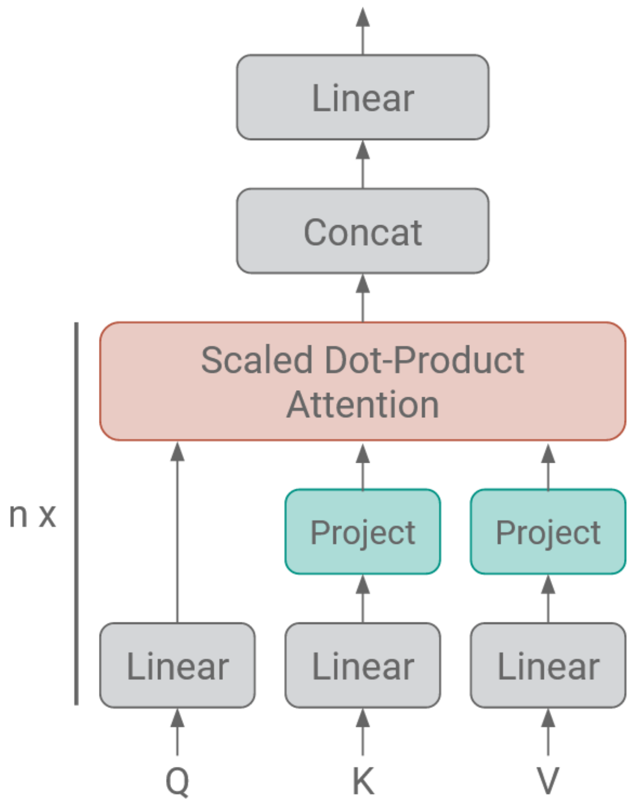

Fig. 6. Linformer self-attention with constant-size projections

4.2.4. *Performer*

The authors of 'Performer: Rethinking attention with transformers' point out that the last two papers rely on sparsity and low-rank assumptions respectively. These assumptions hold in general but are not guaranteed to approximate the full attention procedure up to any precision. The authors noticed the self-attention formula can be implemented more efficiently, not by reducing dimensions but by changing the order of operations. Without the softmax, the formula could be rewritten more efficiently as

$$\text{Attention}(Q, K, V) = Q(K^T V)^T$$

which avoids the calculation of the $N \times N$ matrix. With a clever technique named FAVOR+ they approximate the softmax kernel of the attention matrix, which can be used to change the order of operations as can be seen in Figure 7. This new attention mechanism has backward compatibility, which means it can be directly used on top of an already-trained model, unlike the previously proposed methods [26].

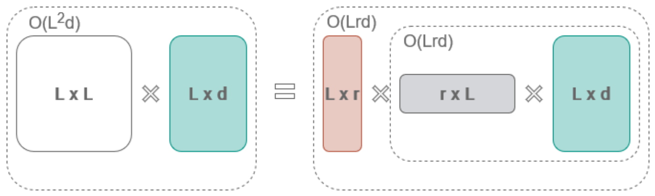

Fig. 7. Performer order of operation

4.2.5. *Sparse Transformer*

The previous section discussed several self-attention approximations. This section will cover several transformer variations that rely on downsampling the problem or slightly adapting it. Downsampling can be achieved by limiting the receptive field of self-attention, similar to how CNNs limit the receptive field from a feedforward layer. The next two sections will provide a short, visual explanation of the modifications made to the attention mechanism.

The sparse transformer proposes a variation upon multi-head attention. Instead of having separate heads that all attend to all available information, they suggest using two different kinds of heads. The first head limits self-attention to windows or chunks of length $\sqrt{n}$. The second head attends to one token in each previous chunk. It should be noted that each token, on average, has $\sqrt{n}/2$ previous tokens to attend to, resulting in an $O(n\sqrt{n})$ complexity. The top row in Figure 8 represents the information the two kinds of heads can access for a given token (darker on the image), while the bottom row depicts the attention matrix (not to scale) of all tokens. The last column depicts a variation of this architecture for data without a periodic structure [27].

Intuitively, this is the middle ground between CNNs and full self-attention. CNNs perform purely locally, and self-attention is fully global. This creates a cross pattern on the input, wherein each layer, information flows inter-chunk and also intra-chunk. By stacking these layers, the hypothesis is that information is able to propagate anywhere. Using chunks of $\sqrt{n}$ strikes the exact balance between the amount of intra- and inter-attention complexity.

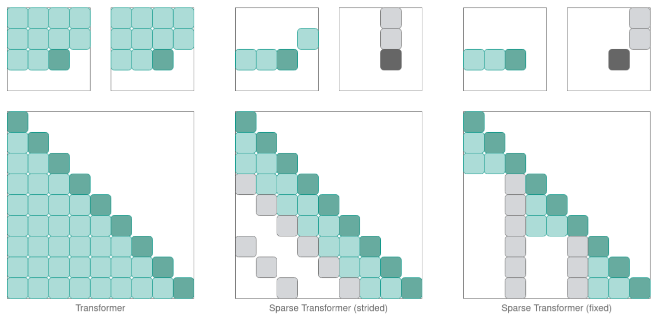

Fig. 8. Sparse Transformer attention patterns

4.2.6. *Big Bird*

Big Bird, the newest entry in the muppet show character models, takes a slightly different approach than the sparse transformer. Instead of settling for a complexity of $n\sqrt{n}$ by using attention chunks of size $\sqrt{n}$, the authors experiment with using constant-size windows for local attention as well as constant-size global attention. They combine this with a linear amount of random attention, which is intended to provide extra context, as seen on the attention matrices in Figure 9. There, the last token attends to the previous token as shown by the windowed attention matrix, to the token before that through the random attention matrix, and to the first two tokens through the global attention matrix. As the complexity of every component is linear, the whole is also linear. It should be noted that, given multiple layers, information can still flow from anywhere to anywhere, but through a constant amount of nodes instead of $\sqrt{n}$, as compared to the sparse transformer. This constant amount of global information is comparable to the Squeeze and Excite networks in the world of CNNs [30].

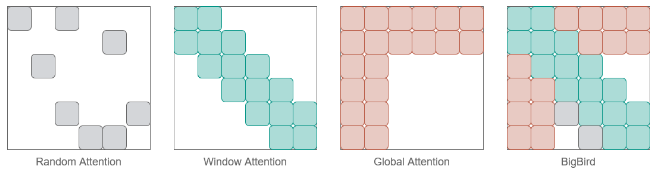

Fig. 9. Bigbird attention patterns

4.2.7. *Perceiver*

Every technique until now has relied on some sort of prior or approximation to reduce complexity. One might wonder whether it is possible to create an attention mechanism with is not approximate and only linear in time. A group of researchers from Deepmind has proposed such an architecture: the Perceiver. They observed that many architectures that are used today, are encoder-only and thus exclusively rely on self-attention. In contrast, the Perceiver architecture is built using a distinct approach, using the decoder part of a transformer to leverage cross-attention, which the authors dubbed 'iterative attention'.

The architecture of the Perceiver is comparable to that of a recurrent neural network (RNN) with a few exceptions. The decoder component of the architecture can be viewed as the latent representation (or internal state), at every time step it performs cross-attention on a transformed version of the input, followed by a self-attention step (similar to the original transformer). The input is provided in full at every step, as opposed to in chunks, as is done in RNNs. By performing all attention operations in this latent representation, it is possible to specify the size of the cross and self-attention, thus eliminating the quadratic complexity. Lastly, unlike an RNN, the weights are not required to be the same across timesteps. These modifications enable the Perceiver architecture to be trained with a greater number of layers than was previously possible for ordinary transformer models. Further details regarding the architecture can be found in Figure 10.

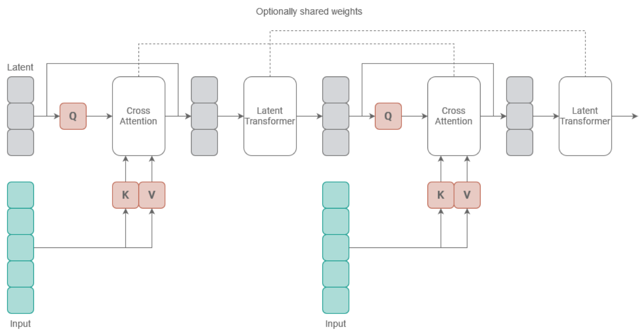

Fig. 10. The perceiver architecture

In the same spirit as the visual transformer (ViT) architecture, the authors of the Perceiver architecture make minimal assumptions about the input data. This results in the model being suitable for a wide range of tasks such as image classification, audio, video, and even point clouds, for which it achieves near state-of-the-art performance [29].

4.2.8. *Remarks*

Every paper in this section makes a compelling case for why quadratic full self-attention is harmful and promotes their own technique as a replacement. However, in practice, this isn't always the case (shocking). These papers often show truncated graphs for small-ish models on limited data, in a way restricting the quadratic transformer model from fully generalizing. Even in some figures in the papers, like Figure 11, it can be noticed (blue line vs green) that the complete transformer continues to learn while the limited solution converges. This is very subtle (certainly without log-scale axes), but even if the standard transformer learns only 10% faster, it is too much of a trade-off for high-accuracy models.

The authors of [31] give an intuitive example of why this is the case. They argue that simple algorithms or certain grammars of greater-than-linear complexity cannot be fully understood by a linear time transformer. The example they give is sorting algorithms $O(nlog(n))$ and Dyck-k languages which are only adequately understood by the full attention transformer. This may seem contrived, but it has merit in large models which try to squeeze the last few percents of accuracy by leveraging these relationships. For very large datasets and/or models, quadratic self-attention will reign supreme for the moment being.

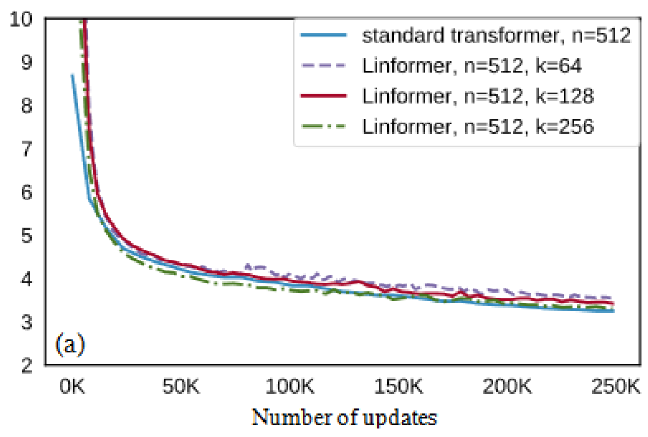

Fig. 11. Transformer vs Linformer performance

5. USE CASES

5.1. GPT & BERT

Vaswani et al. [1] opened the door for transformers in NLP. Two other important works that followed in this field were GPT [16, 32] and BERT [17].

5.1.1. Architecture

GPT uses the decoder architecture we discussed earlier to perform tasks such as classification, multiple-choice question answering, sequence classification, and similarity calculation. The architecture is shown in Figure 12. GPT uses 12 decoder layers, with each multi-head attention layer consisting of 12 heads. The embedding size d_{embed} is 768, while the hidden layer in the feed-forward network is $4\times$ the size of the embedding, and GELU is used as the activation function. A dropout rate of 0.1 is used for residuals, embeddings, and attention. The authors opted to replace the sinusoidal position embeddings with learned position embeddings. Task-specific linear heads are used to process the output of the transformer into predictions.

BERT improved these tasks by using an architecture consisting of encoders instead. The main benefit of this architecture over the use of decoders is the bidirectional attention, allowing the transformer to look at “future” tokens (see Figure 13). Logically, this should be allowed in tasks such as text classification, where the full input can always be used. This is, for example, not the case with sequence generation. There, tokens that have not been generated cannot be used and must be masked during training, making the decoder architecture naturally more suitable.

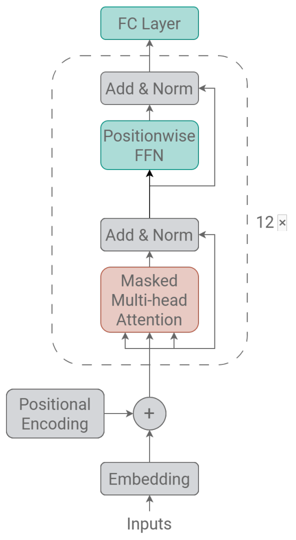

Fig. 12. GPT architecture

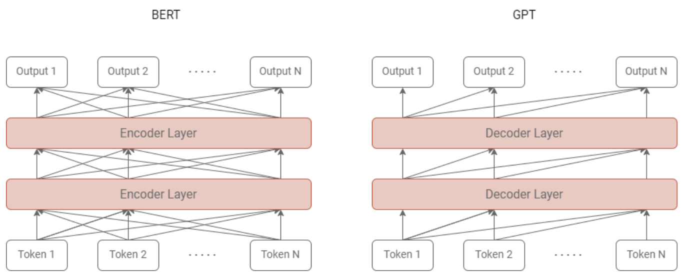

Fig. 13. Comparison of GPT and BERT

BERT was originally created in 2 sizes: $BERT_{BASE}$ and $BERT_{LARGE}$. The former has the same dimensions as GPT, while the latter has more layers (24) a larger d_{embed} (1024), and more attention heads (16).

5.1.2. Training

GPT uses unsupervised pre-training together with supervised (task-specific) fine-tuning. During pre-training, the model learns to predict the next token in a sequence, which does not require any labels. This allows the model to learn powerful token embeddings on large, unlabeled datasets. These embeddings can then be fine-tuned and used for various different tasks.

BERT also employs unsupervised pre-training, but by completing two different tasks. The first task is next sentence prediction (NSP). Given two sequences, the model is tasked to predict if the sequences follow one another. The second objective is called masked language modeling (MLM). Here, tokens are masked within the sequences with a probability of 15%, and the model is asked to predict the masked tokens. However, this would mean the model assumes that unmasked tokens are correct. Instead, we want the model to think critically about all tokens. To achieve this, when a token is chosen for masking (15% chance), it is actually masked with a probability of 80%, replaced with a random token with a probability of 10%, or left untouched with a probability of 10%. Both objectives are trained at the same time, as shown in Figure 14. As with GPT, the model can then be fine-tuned on several different NLP tasks.

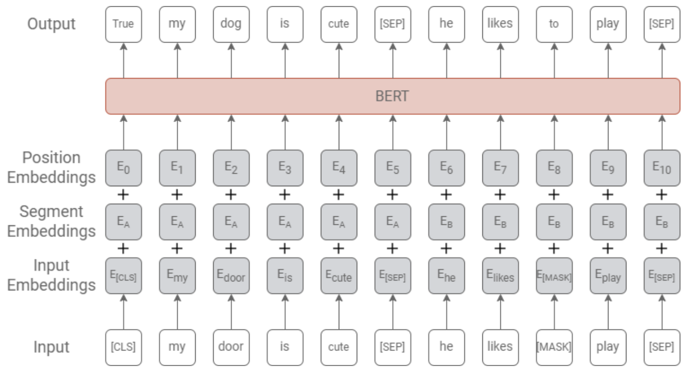
Output
True
my
dog
is
cute
[SEP]
he
likes
to
play
[SEP]
BERT
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Input Embeddings
$E_{[CLS]}$ E_{my} E_{door} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} $E_{[MASK]}$ E_{play} $E_{[SEP]}$
Input
[CLS]
my
door
is
cute
[SEP]
he
likes
[MASK]
play
[SEP]

Fig. 14. BERT training representation. NSP is performed by predicting the true value behind [CLS], while MLM is done by predicting the tokens masked by a [MASK] or replaced by a random token.

5.2. Computer Vision

5.2.1. Breaking Loose of Convolutions

Convolutional neural networks (CNNs) are widely used in various fields but have been particularly influential in computer vision. A convolution is a local operation that aggregates the values of neighboring cells in a tensor using an arbitrary function. This operation is performed for each cell, with some exceptions at the edges [33, 34].

CNNs are prevalent in computer vision due to their locality and translational invariance. Locality refers to the neighbor-based structure of convolutions, which excludes distant data. This is founded on the premise that nearby structures in images are more important than distant ones for classification or detection. Translational invariance refers to the application of the same convolution to each cell, meaning that the position of an object or structure in the image does not matter. These are reasonable simplifications for image-processing models which allow them to perform well. In comparison to fully connected layers, convolution-based architectures have significantly fewer parameters, making them easier to train. This restriction allows these networks to learn quickly, as they are in a sense guided by these limitations [33, 34].

However, these local convolutions may seem too restrictive when only performed once. In these architectures, there are typically tens to hundreds of layers, allowing each cell to eventually gather distant information, or increase its receptive field in technical terms [35].

In practice, this approach can be cumbersome as information may be lost as it passes through multiple operations before being able to gather distant information. It is sometimes necessary to immediately gather distant information to establish complex relationships in image classification, particularly when scaling up already large models that may already possess all the local information. This is where transformers come in, as they can dynamically pay attention to any part of the input based on the current information, enabling them to learn more than their convolution-based counterparts [1].

5.2.2. ViT Architecture

As previously mentioned, the strength of transformers is hindered by their quadratic complexity [1]. If each pixel is treated as a separate token, self-attention becomes computationally expensive, even for small images. For example, a 256x256 image would require at least 256^4 parameters (more than 9 billion) for a single layer. There are two potential solutions to this problem, both of which involve dividing the image into smaller patches. The first is called inter-attention, which encodes a whole patch as a single token and performs attention over the entire image. The second is called intra-attention, which performs attention within each patch and aggregates the information at the end. Both are illustrated on a toy sample in Figure 15 [35, 36].

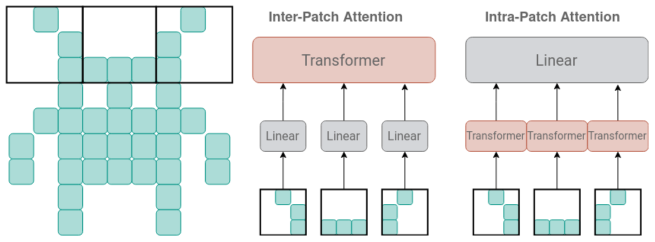

Fig. 15. Comparison of intra- and inter patch attention

The authors opted for the first approach, as the second approach was still implicitly locality-based, which they wanted to avoid. For most models, they used 16x16 patches, and 14x14 patches for the largest model. It is worth noting that smaller patches actually result in more computation because the number of patches increases. These patches are then processed through multiple layers (12 to 32) of multi-head (12 to 16) self-attention before passing through a small multi-layer perceptron head to infer the final class [35].

5.2.3. *Datasets & Details*

The authors used several datasets to train the Vision Transformer models. They use the ILSVRC-2012 ImageNet dataset with 1k classes and 1.3M images, its superset ImageNet-21k with 21k classes and 14M images, and the Google in-house dataset: JFT with 18k classes and 303M high-resolution images. One can notice that these datasets, especially the last dataset, are extremely large. This is required for the transformers to first learn that locality is important and only later learn longer distance relationships [35].

To validate their results and hypotheses, the authors performed an exploratory study into the workings of the model. They noticed that also in images, separate attention heads take on very different roles. Whereas some heads primarily looked at local information, some instantly probed distant pixels as can be seen in Figure 16.

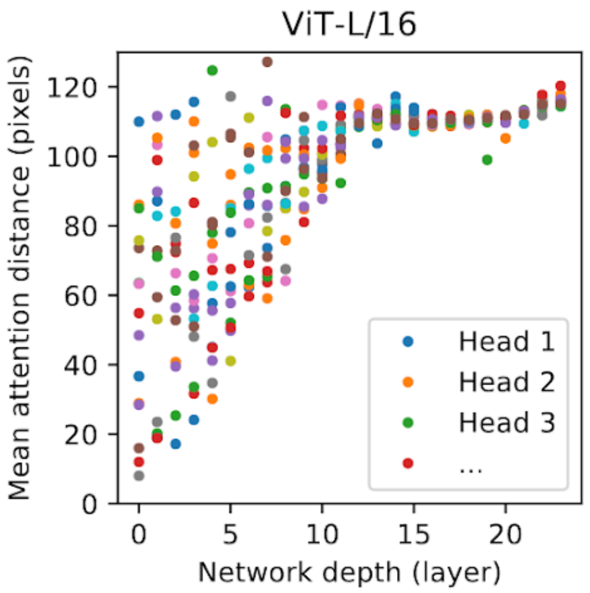

Fig. 16. Mean attention distance by head and depth

5.3. Reinforcement Learning

In reinforcement learning, an agent is trained to perform a certain task in an environment. This is environment is defined by a Markov decision process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, d_0, r, \gamma)$, where $\mathcal{S}$ is the set of states $s \in \mathcal{S}$, $\mathcal{A}$ is the set of actions $a \in \mathcal{A}$, T is the state transition probability distribution $T(s_{t+1}|s_t, a_t)$, d_0 is the initial state distribution, r is the reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and $\gamma \in (0, 1]$ is the discount factor. The goal of the training is to find a policy $\pi(a_t|s_t)$ which maximizes the (discounted) sum of rewards or return G_t.

$$G_t = \sum_{k=0}^{H} \gamma^k r(s_{t+k}, a_{t+k})$$

Where H is the final time step (or horizon), which can be infinite. This problem may seem to be unrelated to the NLP tasks we have thus far discussed, however, there is a convenient way in which we can represent this problem. We can look at the sequence of states, actions, and rewards as a trajectory $\tau = (s_0, a_0, r_0, s_1, ..., s_H, a_H, r_H)$. These trajectories are now comparable to sequences of words, which we know can be processed effectively by transformers.

This method is well-suited for offline reinforcement learning (ORL)[37], shown in Figure 17. In online RL, the agent gets to interact with the environment directly, allowing it to combine exploration and exploitation to find the optimal policy. In contrast, offline RL (or batch RL) does not allow the agent to interact with the environment during training. Instead, the agent learns from a dataset of experiences, such as human demonstrations. This setting is more challenging because the agent cannot adjust the learned policy if it is not accurate. Therefore, offline RL relies on already having access to a dataset of trajectories, which is necessary to train a transformer.

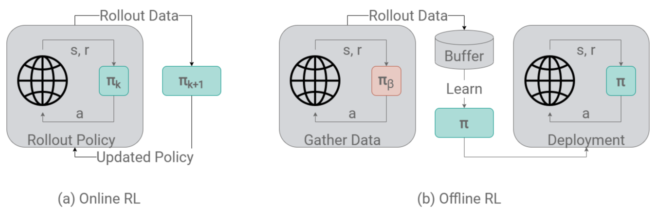

Fig. 17. An illustration of online RL (a) and offline RL (b). In offline RL, the policy needs to be learned from a dataset that is gathered by a different policy, without additional interactions with the environment.

Reinforcement learning is typically done using temporal difference learning. This training process is often unstable due to the “deadly triad” problem [38]. The training process for transformers is closer to that of a classification task. It uses a loss function, which avoids this issue. This idea was originally explored in [39] (Decision Transformer (DT)) and [40] (Trajectory Transformer (TT)) for model-free and model-based RL respectively. These models train by predicting the next token in a sequence of tokens, which is a lot more stable than TD learning (see Figure 18.

5.3.1. *Architecture*

This training procedure effectively results in a behavior cloning model, which performs the same actions as seen in the dataset. Predicting the next state or action, on its own, does not allow the agent to perform exploitation. A simple modification that enables exploitation is to condition the model on the previous states and actions, and the intended outcome.

In the model-free context [39], the model is trained to predict actions in sequences of the form $\tau = (G_0, s_0, a_0, ..., G_H, s_H, a_H)$ (the authors use no discount or $\gamma = 1$ for the calculation of the return). The initial return G_0 is set to the upper bound of the achievable return in order to allow the environment to be exploited by the model, resulting in the selection of actions that lead to that reward.

The approach described in [40] is very similar for model-based RL. Once again, the return is added to the sequence (although they do discount it) and the model is trained to predict the next transition (s_t, a_t, r_t, G_t). Then, a beam search is performed to find the trajectory that leads to the highest return. The authors also mention that it is possible to train the model for a goal-conditioned RL task. This is similar to the return conditioning done in [39], but conditions on the goal state instead of a desired return.

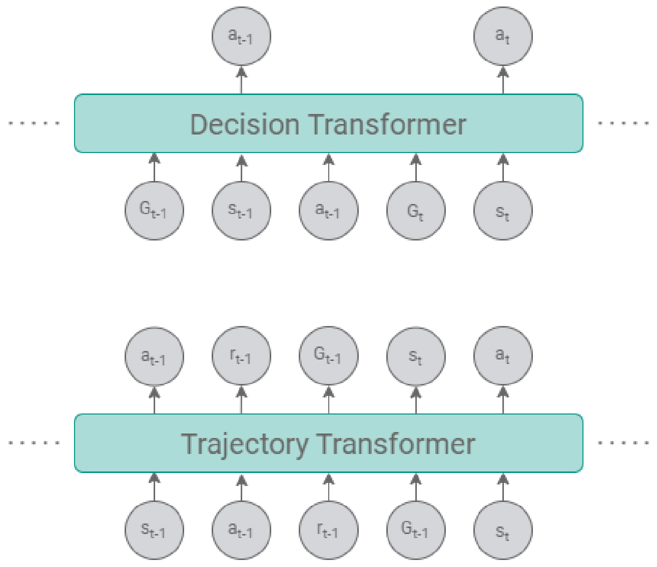

Fig. 18. Decision Transformer [39] and Trajectory Transformer [40] comparison

The transformer architectures that were used in these works were smaller versions of GPT [16]. 1-6 layers and 3-8 attention heads were used, depending on the task.

5.3.2. *Other Related and Future Work*

Transformers have been shown to produce competitive results on D4RL [41] benchmarks for offline RL. One major drawback of these algorithms is the time it takes to train a transformer model. In Table 2, we compare the run time of the Decision Transformer to other SOTA offline reinforcement algorithms. DT takes twice the amount of time to train compared to the next slowest model, despite running for 10× fewer update steps. We also compare performance on the maze2d-large-v1 environment, one that is often not reported in papers. The performance of DT is far below that of the best scoring algorithm. Nevertheless, transformers have shown their potential in this field, and future work may be able to push these architectures to better performance.

Algorithm	Training Time (hours)	Mean Normalized Score
DT[39]	50.6	69.9 ± 22.6
EDAC[42]	6.7	175.2 ± 24.1
SAC-N[42]	5.2	226.9 ± 17.5
IQL[43]	11.5	58.2 ± 8.3
TD3-BC[44]	6.2	104.1 ± 82.4
BC-10%	4.8	5.0 ± 3.8

Table 2. Comparison of offline RL algorithms on the maze2d-large-v1 environment, using the D4RL dataset. All algorithms were run on 1 Nvidia 1080 TI GPU. Implementations were used from the CORL [45] library and using default parameters. Scores are averaged over 3 seeds.

The first and most obvious extension is to adapt the algorithm for online RL. This has been explored more recently by Zheng, Qinqing et al. [46]. As shown in Table 2, there exist environments where DT struggles to perform. Yamagata, Taku et al. [47] aimed to address this by combining Q-learning and transformers. Another goal in RL is to create generalist agents. These are agents which can perform in various environments. Lee, Kuang-Huei et al. build on DT to create a Multi-Game Decision Transformer.

5.4. Other

NLP, Vision, and reinforcement learning are not the only fields in which transformers are used. Unfortunately, it is impossible to cover all use-cases in-depth (or even enumerate them). Therefore this subsection will give a whirlwind overview of some other fields which have profited from transformers or use them in unexpected ways.

5.4.1. *Non-Parametric Transformers*

With the immense progress of neural networks, simpler models are often forgotten. This is not a recommended practice as the knowledge gained from novel techniques can be applied to improve older methods. One class of simpler models that do not rely on learned weights but rather on other data points is k-nearest neighbors (kNN) which only considers its immediate surrounding data points to determine the label. The authors of ”Self-Attention Between Datapoints” challenge the conventional approach of using the features of a single data point to make predictions and propose using all data points. They introduce the Non-Parametric Transformer (NPT) which utilizes self-attention to intelligently examine other data points. Due to this, this architecture has been informally dubbed kNN 2.0. As with other non-parametric techniques, this architecture demonstrates exceptional performance on tabular data and has even achieved state-of-the-art accuracy on several benchmarks, surpassing XGBoost [48].

5.4.2. *Audio*

Transformers are also used for audio understanding. Whisper, recently developed by OpenAI, is an end-to-end speech-processing system, which converts spectrograms directly into text tokens. This model was trained on what the authors call, weakly supervised data, which is not hand-labeled by humans. The authors argue that this trade-off between quality and quantity, which has been successful in image classification and text understanding, should also apply to speech understanding as well [49].

6. DIFFUSION MODELS

The task of generating high-quality images that are indistinguishable from the original distribution has been an active area of research. One of the first successful approaches to this task was the introduction of Generative Adversarial Networks (GANs) in 2014 [8]. GANs consist of a generator and a discriminator, where the generator produces samples that aim to mimic the original distribution, and the discriminator predicts whether a sample is from the original distribution or was generated by the model. By playing a ”game” in which the generator tries to fool the discriminator and the discriminator tries not to be fooled, the model learns how to generate better images.

There have been numerous improvements to the GAN architecture, which will not be covered in this work. One well-known improvement is StyleGAN [10], which is used on the popular website thispersondoesnotexist.com.

6.1. Early Years

Diffusion models take a different approach to image generation. According to the authors of [50], traditional methods for approximating any arbitrary distribution are either incapable of modeling any distribution or must resort to inexact and computationally expensive Monte Carlo techniques. To address this, they propose splitting the task into smaller, easier-to-solve chunks by ”diffusing” (or distorting) the information of a sample slowly, known as the forward diffusion pass. Mathematically, this can be seen as a Markov process as the property $p(x_1, \dots, x_T|x_0) := \prod_{t=1}^{T} p(x_t|x_{t-1})$ holds, indicating that each step is not dependent on any previous state. The training process involves applying the forward pass to all samples, and for each forward step, the model learns the reverse step $p_\theta(x_{t-1}|x_t)$ as seen in Figure 19. To generate a sample that corresponds to the original distribution, the model performs these reverse learned steps repeatedly, starting from random noise, known as the reverse diffusion process. In the context of images, this is achieved by adding noise to the image and learning a model to predict the denoised image. The validity of this method follows from the mathematical fact that applying noise to any image repeatedly will eventually (in the limit) result in completely random noise, but in practice, the process is terminated after a certain number of steps. This cut-off relies on the assumption that the model has learned enough about the decomposition process by this point to be able to recreate the image from pure noise.

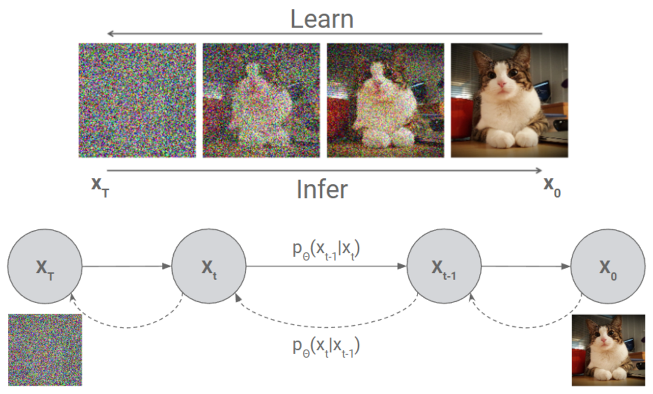

Fig. 19. The forward and reverse diffusion process

6.2. Denoising

The diffusion process for image generation was initially proposed in 2015 but was not widely adopted due to its perceived lack of practicality. In 2020, Ho et al. revisited the idea and proposed improvements, including predicting the noise instead of the image itself, which was found to be easier for neural networks to train. Formally, each step can be defined as $q(x_t|x_{t-1}) := \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$ where β_t is the diffusion speed. The normal distribution is used due to its well-behaved mathematical properties, but many alternatives could be used. They further enhance the model using a modern convolution-based architecture called PixelCNN++, explained in the next section, and sinusoidal embeddings to represent the denoising timestep. These improvements allowed for the approximation of images from the CIFAR-10 dataset, although with a significantly slower speed compared to GANs, requiring 500+ denoising steps [11].

The PixelCNN++ network is designed to handle input vectors with high feature sizes and low channel counts, such as images. These inputs are gradually transformed through pooling layers, which reduce the feature size but increase the channel count, resulting in a compact and global representation of the image. The vectors are then gradually restored to their original size through the use of ”up-convolutions.” To preserve the information in previous layers, residual connections are implemented between network segments of similar size. Additionally, self-attention mechanisms are employed at various layers, further distributing information and enhancing the network’s performance, as demonstrated in Figure 20.

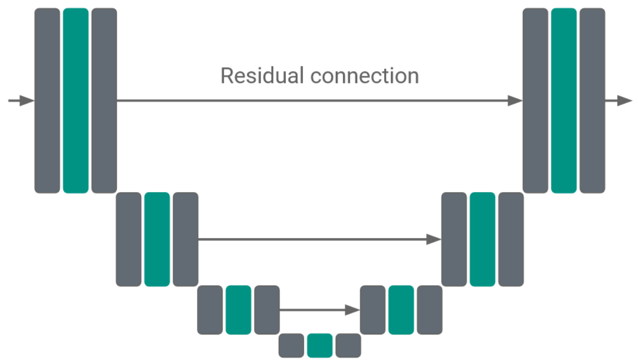

Fig. 20. PixelCNN++ architecture: grey, and green blocks are CNN and self-attention layers respectively

6.3. The Rise of Diffusion Models

Subsequent papers proposed modifications to further enhance the performance of diffusion models in comparison to GANs. Dhariwal and Nichol presented two consecutive papers [51, 13] that aimed to improve the capabilities of diffusion models. They introduced conditional generation techniques, allowing the model to generate images with a specific desired label, such as a specific class in the CIFAR-10 dataset. This is done by adding an embedding of the class label to the input, much like the timestep embedding. This enabled the use of larger datasets for training while maintaining the ability to specify the output. Only adding this embedding did not produce the desired result; the adherence to the requested label seemed lackluster. To solve this they proposed guiding the diffusion process with the aid of an independent classifier to improve the fidelity to the desired label. The predicted noise $\epsilon_\theta(z_\lambda, c)$ is nudged by the gradient of the class (or label) c given the current state $w\sigma_\lambda \nabla_{z_\lambda} log\left(p_\theta(c|z_\lambda)\right)$ where w is a weight factor and σ_λ the standard deviation. This produces images that the model classifies as the desired label with higher certainty, meaning the subject should be more clear. Additionally, the authors recognized that the initial stages of the diffusion process are crucial from a human perspective. They proposed a method that slows the diffusion process at the beginning and speeds it up towards the end, allowing the model to capture finer details in the scene that would be lost using the traditional process.

6.4. Classifier-Free Guidance

One disadvantage of using classifier guidance is the requirement for training an additional model, as ordinary classifiers are not sufficiently powerful due to the presence of noise in the data. Instead, a specialized model is necessary. Ho et al. [52] proposed an alternative method that does not require the use of an additional classifier. They introduced the concept of an unconditional (null) label that is independent of any other label. Then, by performing the diffusion process for both the desired label and the unconditional label. The final output is calculated by adjusting the conditional noise prediction with a specific scale, away from the unconditional noise prediction, such that the label information is increasingly emphasized at each step as can be seen in the formula below. The first part is the conditional prediction, scaled by $1 + w$ and the second is the unconditional prediction, scaled by w where w determines the degree to which the diffuser magnifies the label information.

$$\epsilon_\theta(z_\lambda, c) := (1 + w)\epsilon_\theta(z_\lambda, c) - w\epsilon_\theta(z_\lambda)$$

6.5. CLIP

Neural networks have demonstrated great success in their ability to convert any input into vectors, known as embeddings. When trained effectively, these networks have been shown to separate input based on their semantic meaning, resulting in nearby vectors having similar significance. Typically, these embeddings are utilized for classification into a label or as the next token, but they can also be used directly for performing a nearest neighbor search. This is precisely the approach taken by CLIP’s authors, with a unique twist. They trained the model via contrastive learning on both images and text, with the goal of converting both modalities into a single embedding space. By applying the model to a given text and several images, it is possible to compute the similarity and select the closest image or vice-versa. The architecture consists of two components: the text encoder and the image encoder, both are encoder-only transformers. The image encoder is a ViT model [35], as discussed previously while the text encoder is similar to BERT [17, 53].

6.6. Stable Diffusion

As both diffusion methods and visual-text understanding models became more advanced, they enabled the completion of more complex tasks. Rather than generating images based on static labels, it became viable to use text-image pairs for training and generate images based on any provided text, given that both the language model and the diffuser were trained on a diverse enough range of samples to handle any query in a reasonable way. This sparked a new wave of research, with many new architectures being proposed such as

GLIDE [54] and the original DALL-E [55]; these will not be covered in this work.

Using these foundations, a group from Stability AI set out to create an efficient model and share it with the world. Rombach, Blattmann et al. [14] modified the U-net architecture to incorporate text comprehension into the diffusion process. Adding cross-attention layers to the PixelCNN architecture, which accepts queries from the text and keys and values from the provided or already generated image, enables arbitrary image synthesis. As previously demonstrated, with this attention layer, the model can dynamically select and process important parts of the image and text.

Another improvement involves a change in the space in which the diffusion process is performed. Until this point, the process was conducted in pixel space, meaning that the output of the diffusion process exactly corresponds to the pixel values of the output image. To increase efficiency, the authors introduced an additional step to the process: encoding the image into a compacter latent space, conducting the diffusion process more efficiently within the latent space, and then decoding back to pixel space. This encoding-decoding process is a well-known technique in machine learning, in which the objective is to represent the image in a highly compressed and information-dense space, forcing the network to learn only the important features. This allows for the reconstruction of an image using only limited information.

Performing the diffusion process in this latent space offers several benefits. It is more efficient due to the compact representation of information, and it reduces high-frequency variance in the output. This is because the original diffusion process may alter unimportant pixels in an unnatural way, which would not be noticeable to the classifier and may lead to unstable noise predictions. The decoder is less prone to such effects because it does not rely on these unstable predictions. This paper used a latent space that is 64 times smaller than the original pixel space.

6.7. Datasets

Training these different components requires a significant amount of data, specifically text-image pairs. In the past, large, high-quality datasets were only available to large companies that had the resources to manually classify them. However, due to the flexibility of the model, which does not require an exact label, collecting these pairs has become easier. A small group of individuals decided to take on this task in their spare time, scraping the internet for image-text pairs. These pairs include Instagram pictures with captions, webshop articles with a description, and other common forms of internet content. To ensure the quality of these pairs, they used some filtering based on an older model to ensure that the text is somewhat related to the image. In October 2022, they released the largest open-source image-text dataset to date, LAION-5B, where 5B stands for 5 billion pairs. Since its introduction, this has become the de-facto standard in datasets for large models.

To further improve the quality and flexibility of the dataset, researchers at Laion included useful metadata with each pair: the language of the text, the probability of NSFW, and the aesthetic value. The NSFW probability and aesthetic values are calculated by separate models that were manually trained on a small sample of the data. The dataset is filtered based on the language (English), probability of NSFW ¡ 0.1, and aesthetic ¿ 4.5. The last criterion ensures that the model prefers high-quality artwork or detailed photographs over brochures or other low-quality imagery [56].

6.8. Proprietary models

The final section of this paper focuses on proprietary models developed by major machine-learning labs. While these models will not be explored in depth due to their similarities to each other, it is important to acknowledge the techniques employed to enhance the quality of generated images.

6.8.1. Dall-E 2

One notable example is the Dall-E 2 diffusion model from OpenAI, which is widely recognized as a leading image generator. Dall-E 2 shares many similarities with the Stable Diffusion model, utilizing similar architectural designs and concepts. However, Dall-E 2 operates in pixel space rather than latent space, resulting in a slower model but with slightly improved accuracy. The key distinction between DALL-E 2 and Stable Diffusion is the dataset used for training, which remains undisclosed to the public. However, it is speculated that CLIP was also trained on this same dataset. CLIP is generally favored over OpenCLIP, which is trained on the Lion-5B dataset. The quality and accuracy of the training dataset significantly affect the performance of the model, and it is believed that this is why DALL-E 2 outperforms Stable Diffusion in many instances. Improving the quality of a dataset typically requires costly manual labeling, a significant investment that is only feasible for well-funded research labs.

6.8.2. Imagen

The authors of the Imagen model developed by Google Brain have discovered that scaling the language model (as opposed to the diffuser) results in a significant performance improvement. However, increasing the size of the model also necessitates a larger amount of training data to prevent underfitting. Gathering text-image pairs can be a tedious task. To circumvent this challenge, the authors chose to employ a standard language model, specifically T5, which is comparable to models such as BERT. Acquiring textual data is comparatively easier than obtaining captioned images, thus allowing for larger models. As a result of their immense size, the embeddings produced by these standard text models are even more effective for image generation than those produced by models like CLIP. It is hypothesized that this is because these LLMs have the capability to capture more intricate details from the text and establish more complex relationships within the text than a smaller CLIP model is able to.

6.8.3. eDiff-I

eDiff-I, developed by engineers at NVIDIA, builds upon the concepts of Imagen to further improve image generation quality. The authors observed that during the later stages of generation, the attention towards the text embedding diminishes, resulting in less complete expression of details in the prompt in the generated image (as depicted in Figure 21). To address this issue, two solutions were proposed: utilizing a mixture of experts for different stages in the diffusion process and incorporating both CLIP and T5 as text embeddings. The separation of the reverse diffusion process into distinct stages performed by specialized experts aims to solve the previously mentioned challenge that scaling the diffuser alone does not lead to an improvement in image quality. The combination of CLIP and T5 is intended to leverage the strengths of both models. CLIP is superior at capturing the general information relevant to image generation, while T5 excels at capturing the details. This theory can be extrapolated from the attention patterns of a diffusion model, first relying on CLIP for the main structure and then using T5 information more.

These advancements have led to a substantial increase in the fidelity of the images to the prompt. In complex image synthesis tasks, such as generating a sign with some specific text, the resulting images often depict random symbol sequences. eDiff-I, however, has demonstrated the ability to accurately depict text and capture other nuanced information in the generated images.

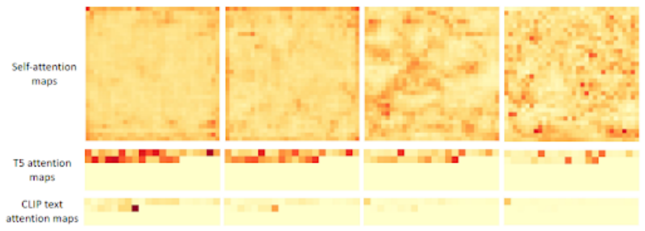

Fig. 21. Attention maps in an ordinary diffusion model for different diffusion steps (left to right)

7. CONCLUSION

In the past few years, every field within machine learning has seen enormous advances. The transformer architecture, originally designed for NLP, has proven to be the holy grail of architectures that can handle any kind of modality. This has sparked lots of research toward improving these models and applying them in various forms.

The effective use of attention mechanisms has played a drastic role in the success of transformer models. Attention allows the model to focus on the most relevant parts of the input, no matter the distance, drastically improving contextual awareness in these models at the cost of quadratic complexity. This advancement has allowed transformers to outperform traditional methods such as RNNs or LSTMs in tasks such as translation, question answering, or text classification.

Efforts have been made toward making transformers more efficient by approximating the full attention mechanism with a linear or close-to-linear complexity. These models have been shown to be competitive with traditional transformers on many tasks but fail to generalize accurately for huge datasets. Conversely, another area that has attracted lots of attention recently is the scaling of transformers to sizes in the order of hundreds of billions. Mixture of Expert models have been proposed as a means to increase the size and expressivity of transformers. Mixture of Models leverage the power of several transformer components, each with its own expertise, which can be selected dynamically to achieve higher accuracy. This is only the start of this research area, little is known about how these huge models learn and behave so it is likely that more novel and innovative solutions will emerge, further boosting the influence of transformers.

The versatility and flexibility of transformers have led to their use in many fields; including computer vision, reinforcement learning, speech recognition, and generative models of many kinds. For all these fields, simply using a transformer architecture combined with a huge dataset has proven to be a sure way to beat the state of the art.

More recently even, transformer-powered diffusion architectures have seen a meteoric rise, providing even more directions for transformer-related research to take place. These architectures take on the task of generating images from a simple textual prompt. Through an innovative combination of ideas, tricks, and lots of transformers, it is now possible to generate breathtaking or humorous images that previously required expert knowledge of photoshop.

In conclusion, the transformer architecture and its variants have had a profound impact on the field of machine learning. Due to its ability to handle a wide range of modalities, the transformer has become a crucial tool for solving any complex problem in machine learning. The future looks bright for transformers and there is no doubt they will continue to play a significant role in shaping the future of machine learning.

8. REFERENCES

[1] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *ArXiv*, vol. abs/1706.03762, 2017.

[2] A. Graves, "Generating sequences with recurrent neural networks," *ArXiv*, vol. abs/1308.0850, 2013.

[3] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. E. Miller, M. Simens, A. Askell, P. Welinder, P. F. Christiano, J. Leike, and R. J. Lowe, "Training language models to follow instructions with human feedback," *ArXiv*, vol. abs/2203.02155, 2022.

[4] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with clip latents," *ArXiv*, vol. abs/2204.06125, 2022.

[5] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," *ACM Computing Surveys*, vol. 55, pp. 1 – 28, 2020.

[6] Midjourney, "Midjourney.com," 2022.

[7] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. L. Denton, S. K. S. Ghasemipour, B. K. Ayan, S. S. Mahdavi, R. G. Lopes, T. Salimans, J. Ho, D. J. Fleet, and M. Norouzi, "Photorealistic text-to-image diffusion models with deep language understanding," *ArXiv*, vol. abs/2205.11487, 2022.

[8] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial nets," in *NIPS*, 2014.

[9] A. Brock, J. Donahue, and K. Simonyan, "Large scale gan training for high fidelity natural image synthesis," *ArXiv*, vol. abs/1809.11096, 2018.

[10] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4396–4405, 2018.

[11] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *ArXiv*, vol. abs/2006.11239, 2020.

[12] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, "Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications," *ArXiv*, vol. abs/1701.05517, 2017.

[13] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," *ArXiv*, vol. abs/2105.05233, 2021.

[14] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10674–10685, 2021.

[15] J. Alammar, "The illustrated transformer," 2018.

[16] A. Radford and K. Narasimhan, "Improving language understanding by generative pre-training," 2018.

[17] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *ArXiv*, vol. abs/1810.04805, 2019.

[18] D. Hendrycks and K. Gimpel, "Bridging nonlinearities and stochastic regularizers with gaussian error linear units," *ArXiv*, vol. abs/1606.08415, 2016.

[19] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[20] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016.

[21] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," 2020.

[22] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," 2021.

[23] R. M. Schmidt, "Recurrent neural networks (rnns): A gentle introduction and overview," 2019.

[24] N. Kitaev, L. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," 2020.

[25] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity," 2020.

[26] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, D. Belanger, L. Colwell, and A. Weller, "Rethinking attention with performers," 2020.

[27] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," 2019.

[28] M. Zaheer, G. Guruganesh, A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed, "Big bird: Transformers for longer sequences," 2020.

[29] A. Jaegle, F. Gimeno, A. Brock, A. Zisserman, O. Vinyals, and J. Carreira, "Perceiver: General perception with iterative attention," 2021.

[30] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-excitation networks," 2017.

[31] G. Weiss, Y. Goldberg, and E. Yahav, "Thinking like transformers," 2021.

[32] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[34] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[35] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *CoRR*, vol. abs/2010.11929, 2020.

[36] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, and A. Ku, "Image transformer," *CoRR*, vol. abs/1802.05751, 2018.

[37] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *ArXiv*, vol. abs/2005.01643, 2020.

[38] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *IEEE Transactions on Neural Networks*, vol. 16, pp. 285–286, 2005.

[39] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," *ArXiv*, vol. abs/2106.01345, 2021.

[40] M. Janner, Q. Li, and S. Levine, "Reinforcement learning as one big sequence modeling problem," in *Neural Information Processing Systems*, 2021.

[41] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, "D4rl: Datasets for deep data-driven reinforcement learning," *ArXiv*, vol. abs/2004.07219, 2020.

[42] G. An, S. Moon, J.-H. Kim, and H. O. Song, "Uncertainty-based offline reinforcement learning with diversified q-ensemble," in *Neural Information Processing Systems*, 2021.

[43] I. Kostrikov, A. Nair, and S. Levine, "Offline reinforcement learning with implicit q-learning," *ArXiv*, vol. abs/2110.06169, 2021.

[44] S. Fujimoto and S. S. Gu, "A minimalist approach to offline reinforcement learning," *ArXiv*, vol. abs/2106.06860, 2021.

[45] D. Tarasov, A. Nikulin, D. Akimov, V. Kurenkov, and S. Kolesnikov, "CORL: Research-oriented deep offline reinforcement learning library," in *3rd Offline RL Workshop: Offline RL as a "Launchpad"*, 2022.

[46] Q. Zheng, A. Zhang, and A. Grover, "Online decision transformer," *ArXiv*, vol. abs/2202.05607, 2022.

[47] T. Yamagata, A. Khalil, and R. Santos-Rodríguez, "Q-learning decision transformer: Leveraging dynamic programming for conditional sequence modelling in offline rl," *ArXiv*, vol. abs/2209.03993, 2022.

[48] J. Kossen, N. Band, C. Lyle, A. N. Gomez, T. Rainforth, and Y. Gal, "Self-attention between datapoints: Going beyond individual input-output pairs in deep learning," 2021.

[49] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," 2022.

[50] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," 2015.

[51] A. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," 2021.

[52] J. Ho and T. Salimans, "Classifier-free diffusion guidance," 2022.

[53] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," 2021.

[54] A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. McGrew, I. Sutskever, and M. Chen, "Glide: Towards photorealistic image generation and editing with text-guided diffusion models," 2021.

[55] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, "Zero-shot text-to-image generation," 2021.

[56] C. Schuhmann, R. Beaumont, R. Vencu, C. Gordon, R. Wightman, M. Cherti, T. Coombes, A. Katta, C. Mullis, M. Wortsman, P. Schramowski, S. Kundurthy, K. Crowson, L. Schmidt, R. Kaczmarczyk, and J. Jitsev, “Laion-5b: An open large-scale dataset for training next generation image-text models,” 2022.

[57] J. Alammar, “The illustrated stable diffusion,” 2022.

[58] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *CoRR*, vol. abs/1409.0473, 2014.

[59] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *NIPS*, 2014.

[60] Z. Kong, W. Ping, J. Huang, K. Zhao, and B. Catanzaro, “Diffwave: A versatile diffusion model for audio synthesis,” 2020.

[61] K.-H. Lee, O. Nachum, M. Yang, L. Y. Lee, D. Freeman, W. Xu, S. Guadarrama, I. S. Fischer, E. Jang, H. Michalewski, and I. Mordatch, “Multi-game decision transformers,” *ArXiv*, vol. abs/2205.15241, 2022.