DeepSeek Improvement on Transformers

Yongge Wang (UNC Charlotte)

January 31, 2025

Abstract

In this paper, we briefly review the improvement that DeepSeek has made on Transformer.

1 Neural networks and Transformers

First we define the ReLU(rectified linear unit) function:

$$\mathrm{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Then a single neuron can be described as follows:

$$f(x) = \mathrm{ReLU}(w_1 x + w_0)$$

Here, the ReLU function introduces non-linearity to the linear expression $w_1 x + w_0$ and w_0 is often referred as the bias. A single neuron with multi-variables can be described as

$$f(\mathbf{x}) = \mathrm{ReLU}(\mathbf{w}^T \cdot (\mathbf{x}, 1))$$

softmax function: The softmax function is a frequently used tool for the normalization of probability distributions. For an N-dimension input vector $\mathbf{x}$, the function transform this vector into a probability distribution, where each probability is directly proportional to the exponential of the corresponding input number. To clarify, prior to applying the softmax function, some components of the input vector may be negative or greater than 1. As a result, the components may not collectively sum up to 1, rendering them unsuitable for interpretation as probabilities. However, after the application of the softmax function, every component of the vector will be confined within the $(0, 1)$ interval, and they will sum up to precisely 1, making them amenable to interpretation as probabilities. Notably, the larger input values will yield correspondingly higher probabilities. More formally, for a vector $\mathbf{x} = (x_1, \cdots, x_N)$, the softmax transformation is defined as

$$\mathrm{softmax}(\mathbf{x}) = (\bar{x}_1, \cdots, \bar{x}_N) \text{ with each } \bar{x}_i \text{ calculated as } \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}.$$

To achieve effective natural language processing (NLP), one crucial objective is the development of a system enabling computers to grasp the meaning of individual words. The significant breakthrough in this realm came from the pioneering concept introduced by Firth [5] in 1957, suggesting that a word's meaning is shaped by the context in which it appears. This idea is often expressed as "a word is characterized by the company it keeps". This is generally accomplished through word embeddings, which involve the mapping of a vocabulary into numerical vectors in the real number space. These real-valued vectors effectively encode the essence of a word in such a way that when two word representations are closer in the vector space, they are expected to share a similar meaning.

The transformer architecture employs the encoder-decoder structure. The encoder takes an input sequence of symbol representations $\mathbf{x} = (x_1, \cdots, x_n)$ and transforms it into a sequence of continuous representations $\mathbf{z} = (z_1, \cdots, z_n)$. Once we have the sequence $\mathbf{z}$, the decoder produces an output sequence $(y_1, \cdots, y_m)$ by generating one symbol at a time. At each step in the decoding process, the decoder incorporates the

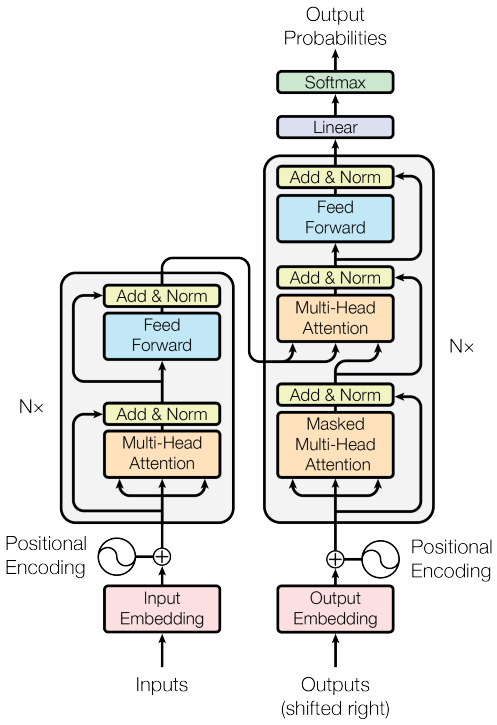

Figure 1: The Transformer - model architecture (from [15])

previously generated symbols as additional input when generating the next symbol. A visual representation of the transformer architecture can be seen in Figure 1, with Nx being 6, indicating that the block labeled Nx is repeated 6 times.

The transformer model, as described in Vaswani et al.'s paper [15], utilizes a word embedding size of $d_{\text{model}} = 512$. In this model, the input sentence $\mathbf{x}$ is initially broken down into a sequence of tokens (words) denoted as $\mathbf{x} = (x_1, \cdots, x_n)$. Each word is associated with a d_{model}-dimensional real-number vector, and this representation is subject to updates during the learning process. As a result, the input sequence can be transformed into a sequence of real-number vectors: $u^{x_1}, \cdots, u^{x_n}$.

To incorporate positional information into the input vector sequence $u^{x_1}, \cdots, u^{x_n}$, we require a series of position vectors. Consider $pos_i = (p_{i,1} \cdots, p_{i,d_{\text{model}}})$, a d_{model}-dimensional real-number vector defined as follows:

$$p_{i,j} = \begin{cases} \sin \frac{i}{10000^{\frac{j}{d_{\text{model}}}}} & \text{if } j \text{ is even} \\ \\ \cos \frac{i}{10000^{\frac{j-1}{d_{\text{model}}}}} & \text{if } j \text{ is odd} \end{cases}$$

The input vectors are augmented with position vectors by letting $\bar{u}^{x_1} = u^{x_1} + pos_1, \cdots, \bar{u}^{x_n} = u^{x_n} + pos_n$. It's important to note that these position vectors remain constant throughout the transformer model's operation and are not updated during the learning process.

The sequence of augmented input vectors, denoted as $\bar{u}^{x_1}, \cdots, \bar{u}^{x_n}$, can be converted into a matrix A with dimensions $n \times d_{\text{model}}$. We can set $Q = A$, $K = A$, and $V = A$ as the query, key, and value matrices, respectively. Then the dot-product based self-attention can be computed as

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

where d_k represents the dimension of the input vectors, which in this case is d_{model}. It's worth mentioning that QK^T forms an $n \times n$ matrix which serves as a representation of the connections between these words.

The matrix QK^T is divided by $\sqrt{d_k}$ and then the softmax is applied to each row. The $n \times d_{\text{model}}$-dimensional attention matrix Attention(Q,K,V) is expected to encode information about each word and its relationships with all other words, with each row capturing such associations. To elaborate, the element at the (i, j) position in the matrix QK^T is determined by $\mathbf{q}_i \cdot \mathbf{k}_j = |\mathbf{q}_i||\mathbf{k}_j| \cos(\theta)$, where $\mathbf{q}_i$ represents the ith row of matrix Q, $\mathbf{k}_j$ is the jth row of matrix K, and θ is the angle between them. In essence, the element at the (i, j) position of the matrix QK^T is larger if and only if the ith word and the jth word are closer in meaning.

Instead of employing a single attention mechanism, the transformer model subdivides the d_{model}-dimensional vector into $h = 8$ segments and employs h separate attention heads. This approach enables us to use individual heads to capture various facets of a word's meaning and its attention to other words simultaneously. Now, let's define d_k as $d_{\text{model}}/h = 512/8 = 64$. For i ranging from 1 to h, we have weight matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$, and $W^O \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ for the query matrix Q, key matrix K, and value matrix V, and output matrix, respectively. These weight matrices are subject to learning during the training process. For each $i = 1, \cdots, 8$, we have

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V).$$

Then we can concatenate these heads to obtain the multi-head attention sub-layer output

$$\text{MultiHead}(Q, K, V) = [\text{head}_1, \cdots, \text{head}_h]\, W^O$$

After the application of the multi-head attention operation to the augmented input vectors $\bar{u}^{x_1}, \cdots, \bar{u}^{x_n}$, we proceed to the "Add & Norm" operation. In this "Add" operation which is also called residual connectoin, the output of the multi-head attention is summed with the input to the multi-head attention operation. In other words, we have $\mathbf{x} + f(\mathbf{x})$, where $f(\mathbf{x})$ represents the multi-head attention operation. This concept was originally introduced in the context of residual learning building blocks by He et al. [6]. For the normalization operation, as described by Ba et al. [1], we first compute two values: $\mu = \frac{\sum_{i=1}^N a_i}{N}$, which is the average of all values a_i in the layer to be normalized, and $\sigma = \sqrt{\frac{\sum_{i=1}^N (a_i - \mu)^2}{N}}$, which is the standard deviation calculated from these values. Then, we normalize each a_i as follows:

$$\bar{a}_i = \frac{a_i - \mu}{\sigma + \varepsilon}$$

where ε is a small added value so that $\bar{a}_i$ are not too large when σ is very small. To reduce the computational cost of re-centering in layer normalization, one can use Root Mean Square Layer Normalization (RMSNorm) as proposed by Zhang and Sennrich [16]. In RMSNorm, the normalization is defined as:

$$\bar{a}_i = \frac{a_i}{\text{RMS(a)}} \qquad \text{where} \quad \text{RMS(a)} = \sqrt{\frac{\sum_{i=1}^N a_i^2}{N}}.$$

RMSNorm is employed in Meta AI's LLaMA model [14], where it is used to normalize the input of each transformer sub-layer instead of the output.

The outcome of the preceding procedure, referred to as the "Add & Norm" operation, serves as the input to a fully connected feed-forward network. This network is individually and consistently applied to every word position, corresponding to each row within the $n \times d_{\text{model}}$ matrix. It involves two linear transformations separated by a ReLU activation function:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

where W_1 is of dimension $d_{\text{model}} \times d_{ff}$ and W_2 is of dimension $d_{ff} \times d_{\text{model}}$. The paper [15] used a parameter of $d_{ff} = 2048$. The FFN's output is directed into the "Add & Norm" operation to derive a sub-layer output, concluding the operations within the Nx box for one iteration. To obtain the final output $\mathbf{z}$ for the Encoder, these operations in the Nx box must be repeated six times.

The output $\mathbf{z}$ from the Encoder serves as an input for the Decoder's "Multi-Head Attention". To elaborate, the sequence $\mathbf{z}$ is transformed into the matrices Q and K, which are subsequently provided as input

to the Decoder's "Multi-Head Attention". The matrix V input for the Decoder's "Multi-Head Attention" is sourced from the "Masked Multi-Head Attention" and the "Add & Norm" operation within the Decoder module. The operations within the Decoder are quite similar to those within the Encoder, with one notable exception being the "Masked Multi-Head Attention". The Decoder enables each position, in other words, each row of the $m \times 512$ dimensional output matrix, to attend to all positions within the Decoder, up to and including that specific position. This constraint is crucial to prevent the flow of information in a leftward direction within the Decoder and thereby preserve its auto-regressive nature. To achieve this, Transformer implement a masking process in the scaled dot-product attention mechanism. During this process, we set all values in the input of the softmax corresponding to illegal connections to $-\infty$.

2 Mixture of Experts (MoE)

One of the techniques utilized in DeepSeek [4] was the incorporation of the Mixture of Experts (MoE) (see, e.g. Baldacchino et al [2]) to replace the single Feed-Forward Neural Network (FFN) within each Transformer layer. In the original Transformer architecture, the FFN processes all positions in the sequence uniformly, which becomes increasingly computationally expensive as the model size scales up.

To address the challenge of scaling models with extremely large parameters while maintaining efficiency, the *Mixture of Experts* (MoE) paradigm was introduced. MoE is a technique that dynamically assigns parts of the model to process different inputs based on their characteristics, effectively partitioning the problem space into homogeneous regions.

An MoE layer consists of multiple "experts" (sub-networks), and only a subset of these experts is activated for any given input. The activation is controlled by a *gating network*, which determines the expert(s) to use for each input token. The output of an MoE layer can be mathematically expressed as:

$$\text{MoE}(x) = \sum_{i=1}^{N} g_i(x) \cdot \text{Expert}_i(x),$$

where N is the total number of experts, $\text{Expert}_i(x)$ represents the i-th expert's computation on input x, and $g_i(x)$ is the gating function, which assigns a weight to each expert based on the input.

The gating function $g(x)$ is typically implemented as a softmax function over the expert scores:

$$g_i(x) = \frac{\exp(s_i(x))}{\sum_{j=1}^{N} \exp(s_j(x))},$$

where $s_i(x)$ is the score assigned to expert i by the gating network. To ensure computational efficiency, only the top k experts (e.g., $k = 1$ or $k = 2$) with the highest scores are activated for each input, reducing the overall computation. The advantages of MoE include:

- Parameter Efficiency: By activating only a subset of experts, the model can scale to billions of parameters without linearly increasing computational costs.
- Specialization: Each expert can specialize in a specific subset of the input space, improving performance on diverse tasks.
- Dynamic Computation: The gating mechanism enables adaptive computation, where different inputs are processed by different parts of the model.

While MoE provides substantial benefits, it also introduces challenges:

- Load Balancing: Ensuring that all experts are utilized equally to prevent some experts from becoming bottlenecks.
- Training Stability: The sparsity and dynamic nature of the gating mechanism can lead to instability during training.
- Memory Overhead: Storing large numbers of experts requires significant memory resources.

To tackle these challenges, DeepSeek [4] introduced specific techniques, including fine-grained expert segmentation, shared expert isolation, and load-balancing mechanisms, to enhance the effectiveness and efficiency of their implementation.

3 Multi-Head Latent Attention

Transformer models rely on Multi-Head Attention (MHA) during generation, but the substantial Key-Value (KV) cache associated with MHA often becomes a bottleneck, limiting inference efficiency. To address this issue, DeepSeek V2 [7] and DeepSeek V3 [8] introduced Multi-Head Latent Attention techniques, which effectively reduce the size of the K, V cache and improve inference performance.

In the transformer multi-head attention (MHA) model, we use three weight matrices: $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$ for the query, key, and value matrices, respectively, and $W^O \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ for the output matrix. Specifically, for each head $i = 1, \cdots, 8$, the attention for head i is defined as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) = \text{softmax}\left(\frac{QW_i^Q(KW_i^K)^T}{\sqrt{d_k}}\right)VW_i^V.$$

To compute the encoding of the j-th output token, we must determine the j-th row of each attention head. Specifically, for the j-th row $\mathbf{o}_j^i$ of head_i, we compute:

$$\mathbf{o}_j^i = \text{softmax}\left(\frac{\mathbf{q}_j W_i^Q(KW_i^K)^T}{\sqrt{d_k}}\right)VW_i^V \tag{1}$$

where $\mathbf{q}_j$ is the j-th row of Q, representing the word embedding of the j-th input token.

To optimize computation, all keys and values must be cached, allowing for efficient reuse. However, this caching requirement incurs significant memory overhead, as the model must store all entries of K and V. During model deployment, this memory burden becomes a critical bottleneck, restricting both the maximum batch size and the processable sequence length.

DeepSeek [7, 8] introduces two key innovations to optimize the size of K, V value caches in the transformer model: *Low-Rank Key-Value Joint Compression* and *Decoupled Rotary Position Embedding (RoPE)*. These advancements aim to reduce memory usage while preserving performance, addressing challenges in scaling transformers to larger tasks.

Low-Rank Key-Value Joint Compression: This method factorizes the key-value (K, V) matrices into low rank matrices. This reduces storage requirements and accelerates inference, while still enabling effective recovery of the original information. We illustrate the Low-Rank Key-Value Joint Compression method with an example. Suppose the word-embedding dimension is $d_{\text{model}} = 4096$ and the latent dimension is $d_c = 1024$. Instead of storing d_{model}-dimensional vectors for the rows of the Key and Value matrices, we use compressed d_c-dimensional representations. To achieve this, we introduce a *down-projection* matrix $W^{DKV} \in \mathbb{R}^{d_{\text{model}} \times d_c}$ and two *up-projection* matrices, W^{UK} and W^{UV}, where $W^{UK}, W^{UV} \in \mathbb{R}^{d_c \times d_k h}$, for reconstructing the keys and values.

For each input token vector $\mathbf{w}_i$, which corresponds to the i-th row of the matrix Q, we compute the compressed representation:

$$\mathbf{c}_i^{KV} = \mathbf{w}_i W^{DKV}$$

This vector $\mathbf{c}_i^{KV}$ serves as the compressed i-th row for both the K and V matrices. As a result, the KV cache requires only ld_c elements instead of ld_{model}, where l represents the number of layers.

During inference, the i-th rows of K and V are reconstructed as:

$$\begin{aligned}(\mathbf{k}_{i,1}^C, \cdots, \mathbf{k}_{i,h}^C) = \mathbf{k}_i^C &= \mathbf{c}_i^{KV} W^{UK}, \\ (\mathbf{v}_{i,1}^C, \cdots, \mathbf{v}_{i,h}^C) = \mathbf{v}_i^C &= \mathbf{c}_i^{KV} W^{UV}.\end{aligned}$$

Notably, we can reformulate equation (1) as:

$$\mathbf{o}_j^i = \text{softmax}\left(\frac{\mathbf{q}_j W_i^Q(KW_i^K)^T}{\sqrt{d_k}}\right)VW_i^V = \text{softmax}\left(\frac{\mathbf{q}_j W_i^Q(K^C W^{UK} W_i^K)^T}{\sqrt{d_k}}\right)V^C W^{UV} W_i^V, \tag{2}$$

where K^C and V^C are the compressed versions of K and V, respectively. This formulation allows W^{UK} to be absorbed into W_i^K and W^{UV} into W_i^V, eliminating the need to explicitly compute the keys and values for attention, thereby reducing computational overhead.

Furthermore, to reduce activation memory during training, DeepSeek employs two matrices, W^{DQ} and W^{UQ}, to compress the query matrix Q from d_{model}-dimensional vectors to d'_c-dimensional representations. That is,

$$\begin{aligned} \mathbf{c}_i^Q &= \mathbf{w}_i W^{DQ}, \\ \mathbf{q}_i^C &= \mathbf{c}_i^Q W^{UQ}. \end{aligned} \tag{3}$$

where $\mathbf{c}_i^Q \in \mathbb{R}^{d'_c}$ is the compressed latent vector for the query $\mathbf{w}_i$, d' denotes the query compression dimension; and $W^{DQ} \in \mathbb{R}^{d_{\text{model}} \times d'_c}$, $W^{UQ} \in \mathbb{R}^{d'_c \times d_{\text{model}}}$.

In summary, a key technique in DeepSeek is the joint approximation of the key and value matrices using matrix factorization, rather than treating them separately. Specifically, we have:

$$\begin{aligned} K &\approx W^{DKV} W^{UK}, \\ V &\approx W^{DKV} W^{UV}. \end{aligned}$$

The low-rank matrices W^{DKV}, W^{UK}, W^{UV} are learned during training through backpropagation. In practical implementaitons, choosing the optimal rank d_c is critical for balancing compression and accuracy. A rank that is too low may lose important information, while a rank that is too high may negate memory savings.

Decoupled Rotary Position Embedding (RoPE): For a query $\mathbf{q}$ and a key $\mathbf{k}$, with their position-coded vectors $\mathbf{q}_m$ (at position m) and $\mathbf{k}_n$ (at position n), Su et al. [12] emphasized that, for an effective position embedding method, the inner product $\mathbf{q}_m \mathbf{k}_n^T$ should be expressible as a function g, which only takes the word embeddings $\mathbf{q}$ and $\mathbf{k}$, along with their relative position $m - n$, as input variables. Specifically, this relationship should be formulated as:

$$\mathbf{q}_m \mathbf{k}_n^T = g(\mathbf{q}, \mathbf{k}, m - n) \tag{4}$$

In contrast, the original Transformer model incorporates positional information by directly adding it to the word embeddings, which does not satisfy the requirement (4).

Su et al. [12] introduced the RoPE, which can be intuitively understood using two-dimensional vectors. For a two-dimensional word-embedding vector, the vector at position m is rotated by an angle of $m\theta$, where θ is a fixed angle. This is equivalent to multiplying the vector by $e^{im\theta}$.

For a general d_{model}-dimensional word-embedding space, the space is divided into $d_{\text{model}}/2$ sub-spaces, and the position encoding is applied to each sub-space. Specifically, for a vector $x = [x_1, x_2, \ldots, x_{d_{\text{model}}}]$, each pair of coordinates (x_{2i}, x_{2i+1}) is rotated as follows:

$$\begin{bmatrix} x'_{2i} \\ x'_{2i+1} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_{2i} \\ x_{2i+1} \end{bmatrix},$$

where θ is determined by the position index. In other words, the positioned vector (x'_{2i}, x'_{2i+1}) is derived by rotating the original vector (x_{2i}, x_{2i+1}) by an angle θ.

If RoPE is directly applied to DeepSeek, it becomes coupled with position-sensitive weight matrices. As a result, W^{UK} can no longer be absorbed into W_i^K because the RoPE matrix corresponding to the currently generated token will lie between W^{UK} and W_i^K, and matrix multiplication is not commutative. This forces the recomputation of the keys for all prefix tokens during inference, leading to inefficiency. To resolve this, the DeepSeek team introduces a decoupled RoPE strategy. This method employs additional multi-head query $\mathbf{q}_{i,j}^R \in \mathbb{R}^{d_h^R}$ and shared key $\mathbf{k}_i^R \in \mathbb{R}^{d_h^R}$ to carry the RoPE information separately from the compressed keys and queries, where d_h^R denotes the per-head dimension of the decoupled queries and key. The decoupled RoPE queries and keys are then concatenated with the up-projected queries and keys.

Let $W^{QR} \in \mathbb{R}^{d'_c \times h d_h^R}$ and $W^{KR} \in \mathbb{R}^{d_{\text{model}} \times d_h^R}$ be the matrices used to produce the decoupled queries and keys, respectively. Additionally, let $\mathbf{c}_i^Q \in \mathbb{R}^{d'_c}$ represent the compressed latent vector corresponding to the encoding $\mathbf{w}_i$ of the i-th token, as defined in (3). Then, we define:

$$
\begin{aligned}
(\mathbf{q}_{i,1}^R, \ldots, \mathbf{q}_{i,h}^R) = \quad & \mathbf{q}_i^R = \mathrm{RoPE}(\mathbf{c}_i^Q W^{QR}) \\
& \mathbf{k}_i^R = \mathrm{RoPE}(\mathbf{w}_i W^{KR}) \\
& \mathbf{q}_{i,j} = [\mathbf{q}_{i,j}^C, \mathbf{q}_{i,j}^R] \\
& \mathbf{k}_{i,j} = [\mathbf{k}_{i,j}^C, \mathbf{k}_i^R]
\end{aligned}
$$

Then in the attention calculation, we compute

$$
\mathbf{q}_{i,j}\mathbf{k}_{i,j}^T = \mathbf{q}_{i,j}^C(\mathbf{k}_{i,j}^C)^T + \mathbf{q}_{i,j}^R(\mathbf{k}_i^R)^T
$$

instead of only $\mathbf{q}_{i,j}^C(\mathbf{k}_{i,j}^C)^T$. It should be noted that, during inference, the decoupled key should also be cached.

The decoupled RoPE approach enhances long-range dependency modeling by separating positional and content information. This separation prevents distortions caused by entangled representations, resulting in more precise long-range attention.

4 KV-Cache saving comparison

In this section, we present a comparative analysis of cache savings using DeepSeek techniques. Our evaluation focuses on ChatGPT 3.5, ChatGPT 4, and ChatGPT 4 Turbo. Table 1 provides an overview of the Transformer parameters for these models, highlighting potential cache savings achieved through multi-head latent attention (MLA).

The parameter estimates for ChatGPT 3.5, ChatGPT 4, and ChatGPT 4 Turbo were obtained using ChatGPT itself; however, their accuracy is not guaranteed☺. It is important to note that both K and V are matrices of size $n \times d_{\text{model}}$, requiring a total storage of $2nd_{\text{model}}n_{\text{layers}}$ for caching. Furthermore, we assume that the real numbers are represented using 32-bits (4 bytes).

By contrast, when utilizing multi-head latent attention, the storage requirement is reduced to nd_cn_{layers} for MLA and an additional nd_{model} elements for the decoupled RoPE scheme.

For reference, NVIDIA's A100 GPU offers either 40GB or 80GB of VRAM, delivering 19.5 TFLOPs (trillion floating-point operations per second) in FP32 mode. However, its GPU memory bandwidth is constrained to approximately 2 TB/s, which can become a limiting factor in memory-intensive computations.

Table 1: KV-value size comparison for GPTs and GPTs with DeepSeek MLA with $d_c = 1024$

Model Name	n	n_{layers}	d_{model}	n_{heads}	KV-size	DeepSeek MLA cache
GPT-3.5	4096	96	4096	32	12.88GB	1.61B
GPT-4	8192	120-128	8192	64	64.40GB	4.00GB
GPT-4 Turbo	128K	128?	8192	64	1.07TB	67.12GB

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Tara Baldacchino, Elizabeth J Cross, Keith Worden, and Jennifer Rowson. Variational bayesian mixture of experts models and sensitivity analysis for nonlinear dynamical systems. *Mechanical Systems and Signal Processing*, 66:178–200, 2016.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[4] Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y Wu, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*, 2024.

[5] John Firth. A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis*, pages 10–32, 1957.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

[8] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[9] Graham Neubig. Neural machine translation and sequence-to-sequence models: A tutorial. *arXiv preprint arXiv:1703.01619*, 2017.

[10] OpenAI. Gpt-4 technical report, 2023.

[11] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[12] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[13] R Thoppilan, D De Freitas, J Hall, N Shazeer, A Kulshreshtha, HT Cheng, A Jin, T Bos, L Baker, Y Du, et al. Lamda: Language models for dialog applications. arxiv 2022. *arXiv preprint arXiv:2201.08239*.

[14] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[16] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.