Architecture of DeepSeek-V3

Molnár Botond

June 4, 2025

Transformer Model Origins

Transformer models were first introduced in the Paper "Attention is all you need" in 2017, which arguably launched the artificial intelligence revolution. It introduced the **self-attention** mechanism which enables the model to understand the whole context of the input without using convolution or sequence based RNNs. [6]

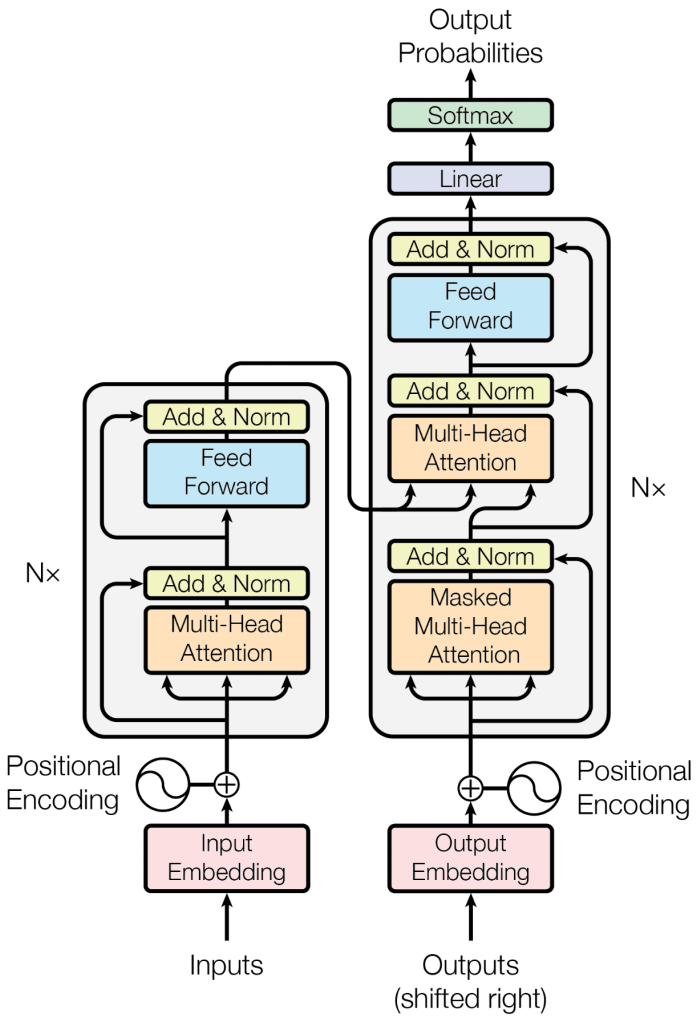
Why not Convolution?

- Convolution is excellent at capturing local patterns, but since they have a limited view on the whole context it falls short of capturing global context. [5]
- The weights of CNNs are constant and learned during training, hence applying the same transformation across the same input. (In transformer models these weights come from the self-attention head $\rightarrow$ calculated dynamically)

Why not Recurrent Neural Networks (RNN)?

- However, the problem of vanishing/exploding gradients has been solved by newer models (LSTM, GRU) they still struggle to maintain "attention". For example, when the model goes through the sequence it might "forget" the early parts of the message or might become insignificant. [3]
- Its sequential processing nature prevents efficient parallelization (the t-th step depends on the $t-1$-th step), which makes it time consuming to train the model. [2]
- It has a fixed sized hidden state (fixed size vector). As the sequence progresses, this vector must continuously update to summarize all relevant past information. For very long sequences, this leads to an information compression problem, where the hidden state cannot retain all the nuances of the entire history. [4]

Transformer Model Architecture



Source: [6]

Input Processing

- The input text must first be converted into numerical representation, usually using byte-pair encoding done on subword level.
- Then each token gets assigned a numerical ID.

Example:

Raw text: *The quick brown fox jumps.*
$\rightarrow$ ["The", "quick", "brown", "fox", "jump", "##s", "."] (## signals the continuation of a word)
$\rightarrow$ [101, 234, 567, 890, 123, 456, 789]

Word Embedding

- **Learned Embeddings**: The transformer uses learned embeddings to convert the input tokens and output tokens to vectors of dimension d_{model}. It learns these representation during the training process.
- **Embedding Table**: $V \times d_{model}$ matrix, where each row represents a unique token in the vocabulary
- **Lookup Process**: After the token enters the embedding layer the model simply "looks up" the corresponding row in this embedding table. The entire row is the vector representation for that token.

Output:

$$E \in \mathbb{R}^{n \times d_{model}}$$

V: vocabulary size
d_{model}: dimension of the model
n: number of tokens

Position Encoding

Since transformer models do not have recurrence or convolution, the order of words is not inherently captured, therefore positional encodings must be used. [6] uses sine and cosine functions to achieve this:

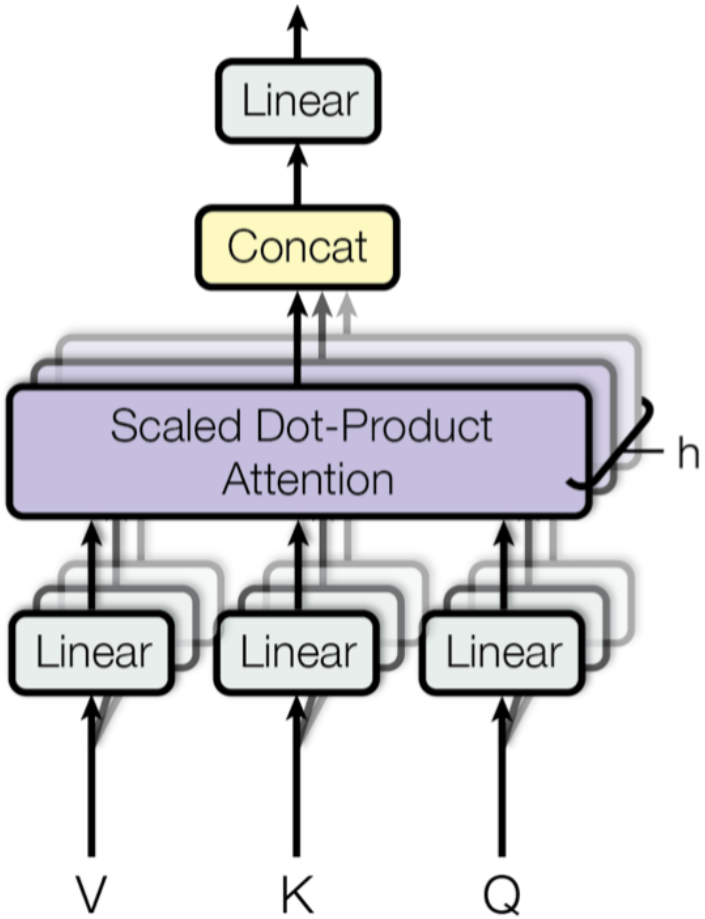
$$PE_{(\text{pos},2i)} = \sin(\text{pos}/10000^{2i/d_{\text{model}}})$$

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

$$Input_{model} = E + EP \in \mathbb{R}^{n \times d_{model}}$$

Where *pos* is the position of the token and i is the dimension.

Encoder Attention Head



Source: [6]

Encoder Attention Head

Each encoder attention head's input consists of the *Query* (Q), *Key* (K), *Value* (V). These values are linearly transformed from the embeddings using learned weights into lower dimensions:

$$Q_i = QW_i^Q$$

$$K_i = KW_i^K$$

$$V_i = VW_i^V$$

Where $Q_i,\ K_i,\ V_i \in \mathbb{R}^{(d_{model}/h)}$,
$W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$
$W_i^K \in \mathbb{R}^{d_{model} \times d_k}$
$W_i^V \in \mathbb{R}^{d_{model} \times d_v}$
h is the number of attention heads.
i is the index of the attention head. [6]

Encoder Attention Head

- **Query (Q)**: Represents what each token attends to
- **Key (K)**: Represents what each token offers as context
- **Value (V)**: The actual content that gets aggregated

Scaled Dot-Product Attention

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

Where
$Q \in \mathbb{R}^{n \times d_k}$,
$K \in \mathbb{R}^{n \times d_k}$,
$V \in \mathbb{R}^{n \times d_v}$.
(In a transformer $d_k = d_v = d_{model}/h$ [6])

Scaled Dot-Product Attention

- $QK^T \in \mathbb{R}^{n \times n}$: Similarity between each *Query* and *Key* is computed. Result of the operation is often called the *"attention scores"*. A large positive dot product indicates high similarity or relevance between the query at position i and the key at position j. This matrix essentially tells us, for each token in the sequence (represented by a query), how much it relates to every other token in the sequence (represented by keys).
- Scaling by $\sqrt{d_k}$ $\left(\frac{QK^T}{\sqrt{d_k}}\right)$: he raw dot products can become very large in magnitude, especially with larger d_{model}. This can push the softmax function into regions where its gradients are extremely small (vanishing gradient). Dividing by $\sqrt{d_k}$ (the square root of the key dimension) normalizes these dot products, preventing them from becoming too large and ensuring more stable gradients for the softmax.

Scaled Dot-Product Attention

- Softmax function ($softmax(\cdot)$): The softmax function converts the scores into a probability distribution. Let $A_{ij} = \frac{(QK^T)_{ij}}{\sqrt{d_k}}$. Then, the softmax is applied row-wise: $\text{softmax}(A_i) = \frac{e^{A_{i,j}}}{\sum_{k=1}^{n} e^{A_{i,k}}}$ for each row i.

  Output of the $softmax(\cdot)$ is the *"attention weight matrix"*: $W \in \mathbb{R}^{n \times n}$.
- Multiplication with the *Value (V)* matrix ($W \cdot V$): This multiplication effectively takes a weighted sum of the *Value* vectors, where the weights are given by the attention weight matrix W. This step aggregates the information from the relevant parts of the sequence into a single context-aware representation for each position.

$$head_j = W \cdot V$$

$$head_j \in \mathbb{R}^{n \times d_v}$$

Where j is the index of the attention head.

Concatenate Outputs From Attention Heads

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \in \mathbb{R}^{n \times d_{model}}$$

where $head_j = Attention(QW_i^Q, KW_i^K, VW_i^Q) \in \mathbb{R}^{n \times (h \cdot d_k)}$,
(since $h \cdot d_k = h \cdot (d_{model}/h) = d_{model}$)
$W^O \in \mathbb{R}^{hd_v \times d_{model}}$: Learned linear projection
n: sequence length

Add and Norm Layer

The output of the Multi-Head Attention is added to its input ($Z^{(l-1)}$) and then layer normalized.

$$Z_{attention\_normalized} = LayerNorm(Z^{(l-1)} + MultiHead(\cdot)) \in \mathbb{R}^{n \times d_{model}}$$

Layer Normalization

For an input vector x (a row in the matrix), it normalizes over the features [1]:

$$\mu = \frac{1}{d_{\text{model}}} \sum_{i=1}^{d_{\text{model}}} x_i \text{ (mean)}$$

$$\sigma^2 = \frac{1}{d_{\text{model}}} \sum_{i=1}^{d_{\text{model}}} (x_i - \mu)^2 \text{ (variance)}$$

$$\text{LayerNorm}(x_i) = \gamma \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

γ and β are learned scaling and shifting parameters, respectively.
ϵ is a small constant for numerical stability (to prevent division by zero).

Position-wise Feed-Forward Network

This sub-layer is applied to each position independently and identically. It consists of two linear transformations with a ReLU activation in between. Let the input be $Z_{attention\_normalized}$

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

- $W_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$ (e.g., 512×2048)
- $b_1 \in \mathbb{R}^{d_{ff}}$
- $max(0, \cdot)$ is the ReLU activation function
- $W_2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$ (e.g., 2048×512)
- $b_2 \in \mathbb{R}^{d_{\text{model}}}$
- Let the output be $FFN_{output} \in \mathbb{R}^{n \times d_{model}} \rightarrow$ This will be passed to another **Add and Norm** sublayer whose output will be passed to the next encoder layer.

Final Encoder Output

After passing through all N encoder layers, the output of the last layer ($Z^{(N)}$) is a sequence of n vectors, each of dimension d_{model}. These vectors are dense, contextualized representations of the input sequence. This final output of the encoder stack is then passed to the decoder's "encoder-decoder attention" mechanism. [6]

Decoder Layer

The decoder in the Transformer model is responsible for generating the output sequence one token at a time, incorporating context from both the input sequence (via the encoder) and the tokens it has already generated. It operates in an auto-regressive fashion.
Each of the N identical layers within the decoder stack is composed of three main sub-layers, each followed by a residual connection and layer normalization.
The main difference to the encoder layer is a new sub-layer, the multi-head self-attention.

Decoder Layer

- Let the input be $Y^{(l-1)}$ for the current decoder layer l. (output from the previous decoder layer or the combined embedding and positional encoding of the partial output sequence for the first layer)
- The encoder's final output $Z_{\text{enc}}^{(N)}$, serves as the memory for the encoder-decoder attention.
- $Y^{(l-1)} \in \mathbb{R}^{m \times d_{model}}$
- m: current length of the partial output sequence being generated
- $Z_{\text{enc}}^{(N)} \in \mathbb{R}^{n \times d_{\text{model}}}$

Masked Multi-Head Self-Attention Sub-layer

1. Input Projections for Q, K, V:

- The input to this sub-layer is $Y^{(l-1)}$.
- For each of the h attention heads ($j = 1, \ldots, h$), $Y^{(l-1)}$ is linearly projected into Query (Q_j), Key (K_j), and Value (V_j) matrices using learned weight matrices $W_j^Q, W_j^K, W_j^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$ (where $d_k = d_v = d_{\text{model}}/h$).
- $Q_j = Y^{(l-1)} W_j^Q$
- $K_j = Y^{(l-1)} W_j^K$
- $V_j = Y^{(l-1)} W_j^V$
- $Q_j, K_j, V_j \in \mathbb{R}^{m \times d_k}$.

Masked Attention

2. Masked Scaled Dot-Product Attention:

- The core attention calculation is performed:

$$\text{head}_j = \text{softmax}\left(\frac{Q_j K_j^T + \text{Mask}}{\sqrt{d_k}}\right) V_j$$

- **The crucial difference here is the "Mask."** This is a look-ahead mask (upper triangular matrix of negative infinities) that is applied to the attention scores before the softmax function.
- It prevents each position in the decoder from attending to subsequent positions. When predicting the i-th token, the mask ensures that the calculation for head_j only considers information from tokens at positions $1, \ldots, i-1$. This is essential for maintaining the auto-regressive property of the decoder, ensuring that predictions for future tokens do not influence current token generation.
- Everithing else works the same as in the encoder layers.
- Output: $\text{head}_j \in \mathbb{R}^{m \times d_k}$.
- $Mask \in \mathbb{R}^{m \times m}$

References I

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

[2] EITCA (European IT Certification Academy). *What are the main challenges faced by RNNs during training and how do Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) address these issues?* 2023. URL: https://eitca.org/artificial-intelligence/eitc-ai-adl-advanced-deep-learning/recurrent-neural-networks-eitc-ai-adl-advanced-deep-learning/sequences-and-recurrent-networks/examination-review-sequences-and-recurrent-networks/what-are-the-main-challenges-faced-by-rnns-during-training-and-how-do-long-short-term-memory-lstm-networks-and-gated-recurrent-units-grus-address-these-issues/ (visited on 06/04/2025).

References II

[3] HeyCoach.in. *Recurrent Neural Networks (RNNs) and their Applications*. 2024. URL: https://blog.heycoach.in/recurrent-neural-networks-rnns-and-their-applications/ (visited on 06/04/2025).

[4] IBM. *Recurrent Neural Networks (RNNs)*. 2023. URL: https://www.ibm.com/think/topics/recurrent-neural-networks (visited on 06/04/2025).

[5] Zhijie Lin et al. "Adaptive RoI-aware network for accurate banknote recognition using natural images". In: *Soft Computing* (May 2025), pp. 1–11. DOI: 10.1007/s00500-025-10649-1.

[6] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).