LLAMA-2, as one of the current mainstream open-source large language models, embodies the core ideas of modern deep learning technology in the field of natural language processing through its complete mathematical formulation. This paper systematically expounds the mathematical expression forms o...

Authors: He Cangping, He Cangping

Date: 2023-08-31T00:00:00+00:00

Abstract

LLAMA has been the most popular open-source large language model in recent months; this paper presents its mathematical formulation.

Full Text

Preamble

Mathematical Principles of the LLAMA-2 Model

He Cangping
cangping@staff.weibo.com
WEIBO.COM

Abstract

LLAMA has emerged as the most popular open-source Large Language Model (LLM) in recent months. This paper presents its mathematical formulation in comprehensive detail.

Keywords: LLAMA, Large Language Model, LLM

The release of ChatGPT has propelled large language models to the forefront of artificial intelligence research. The open-source LLAMA model [**?**] achieves performance metrics comparable to ChatGPT, serving as the foundation for

numerous institutions developing large-scale models. For instance, Baichuan-7B[1] adopts the same architecture as LLAMA. On July 18, 2023, LLAMA-2 [**?**] was released, maintaining the original model architecture while optimizing code performance and permitting commercial use.

To facilitate both rapid business application and rigorous theoretical research, this paper provides the mathematical formulation of the LLAMA model, translating program code into mathematical notation. Where discrepancies exist between the code and the original paper, the implementation in the code shall prevail.

Completion date: July 31, 2023

2 Function Definitions

As preliminary groundwork, this section defines several fundamental functions. Current PyTorch implementations adopt row-major array ordering with zero-based indexing; consequently, all vectors and matrices in this paper follow row-major conventions, with element indices starting at zero.

For any positive integers m and n, row vectors are denoted by bold lowercase letters in the form $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$. Matrices are denoted by uppercase letters. The softmax function is defined as:

$$\operatorname{smax}(\mathbf{x}) = \frac{(e^{x_0}, e^{x_1}, \ldots, e^{x_{n-1}})}{\sum_{i=0}^{n-1} e^{x_i}}$$

For matrices, softmax operates row-wise:

$$\operatorname{smax}(X) = \begin{pmatrix} \operatorname{smax}(x_{0:}) \\ \operatorname{smax}(x_{1:}) \\ \vdots \\ \operatorname{smax}(x_{m-1:}) \end{pmatrix}$$

where $x_{i:} = (x_{i0}, x_{i1}, \ldots, x_{i,n-1})$, and semicolons within parentheses denote row separation.

Element-wise operations extend to logarithms and the silu activation function. The logarithm of a vector or matrix applies the logarithm to each element:

$$\log(\mathbf{x}) = (\log(x_0), \log(x_1), \ldots, \log(x_{n-1}))$$

For real numbers x, the silu activation function is:

$$\operatorname{silu}(x) = \frac{x}{1 + e^{-x}}$$

[1] https://github.com/baichuan-inc/baichuan-7B

The silu function applies element-wise to vectors and matrices. Figure 1 illustrates the silu activation function.

Root Mean Square Layer Normalization (RMSNorm) is defined as:

$$\text{rnor}(X) = \begin{pmatrix} \frac{x_{00}}{\sigma_0} & \frac{x_{01}}{\sigma_0} & \cdots & \frac{x_{0,n-1}}{\sigma_0} \\ \frac{x_{10}}{\sigma_1} & \frac{x_{11}}{\sigma_1} & \cdots & \frac{x_{1,n-1}}{\sigma_1} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{x_{m-1,0}}{\sigma_{m-1}} & \frac{x_{m-1,1}}{\sigma_{m-1}} & \cdots & \frac{x_{m-1,n-1}}{\sigma_{m-1}} \end{pmatrix}$$

where $\sigma_i = \sqrt{\frac{1}{n}\sum_{j=0}^{n-1} x_{ij}^2}$ for $i = 0, 1, \ldots, m-1$.

Given a row vector $\hat{\mathbf{x}} = (\hat{x}_0, \hat{x}_1, \ldots, \hat{x}_{n-1})$, row-wise addition with a matrix is defined as:

$$X + \hat{\mathbf{x}} = \begin{pmatrix} x_{00} + \hat{x}_0 & x_{01} + \hat{x}_1 & \cdots & x_{0,n-1} + \hat{x}_{n-1} \\ x_{10} + \hat{x}_0 & x_{11} + \hat{x}_1 & \cdots & x_{1,n-1} + \hat{x}_{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m-1,0} + \hat{x}_0 & x_{m-1,1} + \hat{x}_1 & \cdots & x_{m-1,n-1} + \hat{x}_{n-1} \end{pmatrix}$$

Rotary Position Embeddings (RoPE) [**?**] constitute a common component in large language models, designed to "implement relative position encoding through absolute position encoding." Detailed derivations are available on the designer's website[2] and in [**?**].

For any even integer $n_6 \geq 2$ and token sequence length n_3, for all $i = 0, 1, \ldots, n_3 - 1$, the rotation matrix A_i is:

$$A_i = \begin{pmatrix} \cos i\bar{\theta}_0 & -\sin i\bar{\theta}_0 & 0 & 0 & \cdots & 0 & 0 \\ \sin i\bar{\theta}_0 & \cos i\bar{\theta}_0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos i\bar{\theta}_2 & -\sin i\bar{\theta}_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin i\bar{\theta}_2 & \cos i\bar{\theta}_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos i\bar{\theta}_{n_6-2} & -\sin i\bar{\theta}_{n_6-2} \\ 0 & 0 & 0 & 0 & \cdots & \sin i\bar{\theta}_{n_6-2} & \cos i\bar{\theta}_{n_6-2} \end{pmatrix}$$

Matrix A_i has dimensions $n_6 \times n_6$, typically 128×128. For radian values $\bar{\theta}_t$ where $t = 0, 2, 6, \ldots, n_6 - 2$, the original paper [**?**] uses fixed values $\bar{\theta}_t = 10000^{-t/n_6}$.

For any real row vector $\mathbf{x} = (x_0, x_1, \ldots, x_{n_6-1})$ and non-negative integer i, the rotary function is defined as:

[2] https://kexue.fm/archives/8265

$$\text{rope}(\mathbf{x}, i) = \mathbf{x}A_i$$

Leveraging the pattern of elements in matrix A_i, the matrix-vector multiplication in Equation (1) can be reformulated as element-wise vector multiplication to reduce computational cost:

$$\text{rope}(\mathbf{x}, i) = \mathbf{x} \otimes \xi_1 + (-x_1, x_0, -x_3, x_2, \dots, -x_{n_6-1}, x_{n_6-2}) \otimes \xi_2$$

where $\otimes$ denotes element-wise multiplication, $\xi_1 = (\cos i\bar{\theta}_0, \cos i\bar{\theta}_0, \cos i\bar{\theta}_2, \cos i\bar{\theta}_2, \dots, \cos i\bar{\theta}_{n_6-2}, \cos i\bar{\theta}_{n_6-2})$, and $\xi_2 = (\sin i\bar{\theta}_0, \sin i\bar{\theta}_0, \sin i\bar{\theta}_2, \sin i\bar{\theta}_2, \dots, \sin i\bar{\theta}_{n_6-2}, \sin i\bar{\theta}_{n_6-2})$.

For a matrix X of size $n_3 \times n_6$, row-wise rotation is defined as:

$$\text{rope}(X, i) = \begin{pmatrix} \text{rope}(x_{0:}, 0) \\ \text{rope}(x_{1:}, 1) \\ \vdots \\ \text{rope}(x_{n_3-1:}, n_3 - 1) \end{pmatrix} = (\text{rope}(x_{0:}, 0); \text{rope}(x_{1:}, 1); \dots; \text{rope}(x_{n_3-1:}, n_3-1))$$

3 Complete View of the LLAMA Model

We define several constants with their typical values. These typical values represent one parameter configuration from Meta's pretrained models, specifically LLAMA-2-7B; other configurations are available on the LLAMA source code website[3]. Let n_1 denote the vocabulary size (number of tokens), typically 32,000; n_2 the token embedding dimension, typically 4,096; n_3 the input sequence length, a positive integer specified before entering the model and remaining constant thereafter; n_5 the number of self-attention heads, typically 32; n_6 the per-head dimension, equal to n_2/n_5, typically 128; n_7 the feed-forward network width, typically 11,008; and n_8 the number of decoder layers, typically 32. Note that these constants share the same meanings as those with identical names in [**?**].

Figure 2 illustrates the complete LLAMA model architecture. The model input is a token sequence in the form:

$$\text{<s>␣token}_1\text{␣token}_2\text{␣token}_3\text{␣}\cdots\text{␣token}_{n_3-2}\text{␣token}_{n_3-1}$$

where ␣ represents explicit whitespace separating tokens. Token 0 is always , marking the sequence beginning. For example, the sentence "Who is the 45th President of the United States?" corresponds to the token sequence:

<s>␣Who␣is␣the␣4␣5␣th␣President␣of␣the␣United␣States␣?

[3]https://huggingface.co/meta-llama

Upon entering LLAMA, tokens are immediately converted to vectors. Consequently, the token sequence transforms into matrix Z_0, where each row vector corresponds to one token. Matrix Z_0 is then fed to decoder layer 0, which outputs matrix Z_1. This matrix subsequently feeds into decoder layer 1, and the process continues iteratively. The output from decoder layer $n_8 - 1$ is $\bar{Z}_{n_8}$, which constitutes the final LLAMA model output.

4 Creating Input Sequences

An input sequence can be any specified text segment, which is then converted into a token sequence using methods such as Byte Pair Encoding, WordPiece, or Unigram Language Model. Let the token vocabulary be denoted as $C = \{c_0, c_1, \ldots, c_{n_1-1}\}$, containing several special tokens: , , and , representing undefined, sequence start, and sequence end, respectively. For Chinese, tokens are individual characters, punctuation marks, or byte representations of characters. For instance, "你" and "好" correspond to tokens "你" and "好", while "啊" splits into three tokens: <0xE5>␣<0x95>␣<0x8A>. For English, tokens are subword fragments; any word can be segmented into multiple tokens, such as "unaffable" splitting into "una␣ff␣able".

After converting all Chinese and English text in the input to tokens, we obtain the tokenized input sequence. All subsequent references to input sequences denote this tokenized form.

Each token c_i in the vocabulary is embedded into a row vector $\mathbf{d}_i$ of size $1 \times n_2$, typically 1×4096. Stacking these row vectors sequentially forms matrix $D = (\mathbf{d}_1; \mathbf{d}_2; \ldots; \mathbf{d}_{n_1})$ with dimensions $n_1 \times n_2$, typically 32000×4096.

Let the input sequence be denoted as $\tau = \tau_0 \tau_1 \ldots \tau_{n_3-1}$, where $\tau_i \in C$ for $i = 0, 1, \ldots, n_3 - 1$. The positional indices of tokens in τ are recorded as $\mathbf{t} = (t_0, t_1, \ldots, t_{n_3-1})$, and their vocabulary indices as $\hat{\mathbf{t}} = (\hat{t}_0, \hat{t}_1, \ldots, \hat{t}_{n_3-1})$, where $t_i \in \{0, 1, \ldots, n_3 - 1\}$ and $\hat{t}_i \in \{0, 1, \ldots, n_1 - 1\}$. For the example sequence:

<s>␣Who␣is␣the␣4␣5␣th␣President␣of the␣United␣States␣?

the positional encoding is $\mathbf{t} = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)$, and the vocabulary indices are $\hat{\mathbf{t}} = (1, 11644, 338, 278, 29871, 29946, 29955, 386, 7178, 310, 278, 3303, 3900, 29973)$.

5 Mask Matrix

The mask matrix M has dimensions $n_3 \times n_3$, typically 128×128. Elements on the main diagonal and lower triangle equal 0, while upper triangle elements equal $-\infty$. In practice, $-\infty$ is represented as the smallest value supported by the data type (e.g., -3.40282×10^{38} for torch.float32, -65504 for torch.float16).

6 Decoder

LLAMA model inputs are token sequences that cannot be directly processed through matrix or vector operations. They must first be converted to matrix form Z_0, which occurs before decoder layer 0. Given input sequence $\tau = \tau_0\tau_1 ... \tau_{n_3-1}$, for each $i = 0, 1, ..., n_3 - 1$, we extract row $\hat{t}_i$ from matrix D and place it at row t_i of matrix Z_0. Matrix Z_0 has dimensions $n_3 \times n_2$, typically $n_3 \times 4096$.

All sublayers described in this section belong to decoder layer 0; this will not be restated for each sublayer.

6.1 Layer Normalization Sublayer

Matrix $\bar{Z}_0$ has dimensions $n_3 \times n_2$, typically $n_3 \times 4096$, and is computed as:

$$\bar{Z}_0 = \text{rnor}(Z_0)$$

6.2 Self-Attention Sublayer

We introduce the query weight matrix W_0^1 with dimensions $n_2 \times n_2$, typically 4096×4096 (superscript 0 indicates decoder layer 0). Denoting $(W_0^1)^T$ as W_0^{1T}, the query matrix is:

$$Q_0 = \bar{Z}_0 W_0^{1T}$$

with dimensions $n_3 \times n_2$, typically $n_3 \times 4096$.

The key weight matrix W_0^2 (dimensions $n_2 \times n_2$) yields the key matrix:

$$K_0 = \bar{Z}_0 W_0^{2T}$$

Similarly, the value weight matrix W_0^3 produces the value matrix:

$$V_0 = \bar{Z}_0 W_0^{3T}$$

Both K_0 and V_0 have dimensions $n_3 \times n_2$.

The query matrix Q_0 is split column-wise into n_5 blocks $Q_{0,i}$ for $i = 0, 1, ..., n_5 - 1$:

$$Q_0 = [Q_{0,0}, Q_{0,1}, ..., Q_{0,n_5-1}]$$

Each $Q_{0,i}$ has dimensions $n_3 \times n_6$, typically $n_3 \times 128$. These blocks are rotated: $\bar{Q}_{0,i} = \text{rope}(Q_{0,i})$, preserving dimensions.

Analogously, K_0 splits into blocks $K_{0,i}$ that are rotated to $\bar{K}_{0,i} = \text{rope}(K_{0,i})$, and V_0 splits into blocks $V_{0,i}$.

We compute:

$$U_{0,i} = \text{smax}\left(\frac{\bar{Q}_{0,i}(\bar{K}_{0,i})^T}{\sqrt{n_6}} + M\right) V_{0,i}$$

with dimensions $n_3 \times n_6$. The n_5 submatrices are concatenated row-wise:

$$U_0 = [U_{0,0}, U_{0,1}, \ldots, U_{0,n_5-1}]$$

forming a matrix of size $n_3 \times n_2$, typically $n_3 \times 4096$.

The output weight matrix W_0^4 (dimensions $n_2 \times n_2$) produces the output matrix:

$$R_0 = U_0 W_0^{4T}$$

with dimensions $n_3 \times n_2$.

6.3 Residual Sublayer

The residual connection computes:

$$Y_0 = Z_0 + R_0$$

with dimensions $n_3 \times n_2$, typically $n_3 \times 4096$.

6.4 Feed-Forward Sublayer

First, we apply layer normalization:

$$\bar{Y}_0 = \text{rnor}(Y_0)$$

with dimensions $n_3 \times n_2$.

We introduce the gate weight matrix W_0^5 (dimensions $n_7 \times n_2$, typically 11008×4096), the down weight matrix W_0^6 (dimensions $n_2 \times n_7$, typically 4096×11008), and the up weight matrix W_0^7 (dimensions $n_7 \times n_2$, typically 11008×4096). The feed-forward computation is:

$$\hat{Y}_0 = \text{silu}(\bar{Y}_0 W_0^{5T}) \otimes (\bar{Y}_0 W_0^{7T}) W_0^{6T}$$

where $\otimes$ denotes element-wise multiplication. The resulting matrix has dimensions $n_3 \times n_2$, typically $n_3 \times 4096$.

Decoder layer 0's final output is:

$$Z_1 = Y_0 + \hat{Y}_0$$

with dimensions $n_3 \times n_2$, typically $n_3 \times 4096$.

6.5 Decoder Stacking

Decoder layer 0 receives matrix Z_0 and outputs Z_1. All decoder layers perform identical computations: layer 1 takes Z_1 as input and outputs Z_2, and so forth. Decoder layer $n_8 - 1$ receives Z_{n_8-1} and outputs Z_{n_8}. For $j = 0, 1, \ldots, n_8 - 1$, each matrix Z_j has dimensions $n_3 \times n_2$, typically $n_3 \times 4096$.

6.6 Model Output

The final LLAMA model output is:

$$\bar{Z}_{n_8} = \text{rnor}(Z_{n_8})$$

with dimensions $n_3 \times n_2$, typically $n_3 \times 4096$.

7 Generating the Next Token

We introduce weight matrix $\bar{W}_8$ with dimensions $n_1 \times n_2$, typically 32000×4096. The logits matrix is:

$$H = \bar{Z}_{n_8} \bar{W}_8^T$$

with dimensions $n_3 \times n_1$, typically $n_3 \times 32000$. Extracting the last row (row $n_3 - 1$) yields vector ϕ of length n_1, typically 32000. This vector $\phi = (\phi_0, \phi_1, \ldots, \phi_{n_1-1})$ represents the logits for the next token.

To introduce randomness and reduce repetition, various scoring modifications may be applied to ϕ before token generation.

7.1 Scoring Modification: Repetition Penalty

The repetition penalty coefficient $\alpha_1 > 0$ is a hyperparameter that scales elements of ϕ at positions specified by $\hat{\mathbf{t}} = (\hat{t}_0, \hat{t}_1, \ldots, \hat{t}_{n_3-1})$. For each $i \in \{\hat{t}_0, \hat{t}_1, \ldots, \hat{t}_{n_3-1}\}$:

$$\bar{\phi}_i = \begin{cases} \alpha_1 \phi_i & \text{if } \phi_i < 0 \\ \phi_i / \alpha_1 & \text{if } \phi_i \geq 0 \end{cases}$$

The modified values $\bar{\phi}_i$ replace the corresponding elements in ϕ.

7.2 Scoring Modification: Temperature

Temperature $\alpha_2 > 0$ is a hyperparameter that scales logits: $\bar{\phi} = \phi/\alpha_2$. For narrative convenience, we continue to denote the modified scores as ϕ.

7.3 Probability Sampling

Let $\widehat{\phi} = \text{smax}(\phi)$, a vector of length n_1, typically 32000. We randomly select an integer from $\{0, 1, 2, \ldots, n_1 - 1\}$, where integer i is chosen with probability $\widehat{\phi}_i$. The selected index is denoted $\widehat{t}_{n_3}$, and the corresponding token is τ_{n_3}, which is appended to the input sequence τ. Generation terminates if τ_{n_3} equals or if sequence τ reaches the specified length; otherwise, generation continues.

8 Fine-Tuning Training

Given any sentence, a tokenizer segments it into token sequence $\tau = \tau_0\tau_1 \ldots \tau_{n_3-1}$ of length n_3, where $\tau_i \in C$. The vocabulary indices are $\hat{\mathbf{t}} = (\hat{t}_0, \hat{t}_1, \ldots, \hat{t}_{n_3-1})$. For example:

Who is the 45th President of the United States? Donald Trump.

tokenizes as:

<s>␣Who␣is␣the␣4␣5␣th␣President␣of␣the␣United␣States␣?␣Donald␣Trump␣.

denoted τ^1 with length $n_3 = 17$ and vocabulary indices:

$$\hat{\mathbf{t}}^1 = (1, 11644, 338, 278, 29871, 29946, 29955, 386, 7178, 310, 278, 3303, 3900, 29973, 18935, 27504, 29889)$$

Feeding sequence τ into the model yields logits matrix H of dimensions $n_3 \times n_1$, typically $n_3 \times 32000$. Removing the last row of H produces a new logits matrix $\bar{H}$ of dimensions $(n_3 - 1) \times n_1$, typically $(n_3 - 1) \times 32000$. For input sequence τ^1, $\bar{H}$ has dimensions 16×32000.

8.1 Fine-Tuning Format 1

We extract the last $n_3 - 1$ elements of $\hat{\mathbf{t}}$ to form $\bar{\mathbf{t}} = (\bar{t}_0, \bar{t}_1, \ldots, \bar{t}_{n_3-2}) = (\hat{t}_1, \hat{t}_2, \ldots, \hat{t}_{n_3-1})$, of length $n_3 - 1$. For τ^1:

$$\bar{\mathbf{t}}^1 = (11644, 338, 278, 29871, 29946, 29955, 386, 7178, 310, 278, 3303, 3900, 29973, 18935, 27504, 29889)$$

8.2 Fine-Tuning Format 2

For question-answering tasks, question token positions can be masked with (value -100 in LLAMA). For τ^1, this yields:

$$\bar{\mathbf{t}}^1 = (-100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, 18935, 27504, 29889)$$

where the final three non-masked values correspond to the answer "Donald Trump."

8.3 Cross-Entropy Loss

Let $\tilde{H} = \text{smax}(\bar{H})$, with dimensions $(n_3 - 1) \times n_1$, typically $(n_3 - 1) \times 32000$. For τ^1, $\tilde{H}$ has dimensions 16×32000.

The loss function is:

$$L = -\sum_{\substack{i=0 \\ \bar{t}i \neq -100}}^{n_3-2} \log(\tilde{h}_{i,\bar{t}i})$$

Row i of $\tilde{H}$ represents the probability distribution for generating token $i + 2$ based on the first $i + 1$ tokens, while $\bar{t}_i$ is the index of the actual token at position $i + 2$ in the input sequence. Training minimizes this loss, making model-generated tokens match the input sequence.

9 Terminology Mapping

All mathematical formulas in this paper are extracted from the LLAMA model source code. For clarity, we map code objects to mathematical symbols and typical values:

- **vocab_{size}**: Vocabulary size, n_1, 32000
- **hidden_{size}**: Token embedding dimension, n_2, 4096
- **seq_{len}**: Token sequence length, n_3, increments by 1 each iteration; maximum 2049 for LLAMA-7B, 4096 for LLAMA-2-7B
- **num_{{{attention}}_{{heads}}}**: Number of self-attention heads, n_5, 32
- **head_{dim}**: Per-head dimension, n_6, 128
- **intermediate_{size}**: Feed-forward network width, n_7, 11008
- **num_{{{hidden}}_{{layers}}}**: Number of decoder layers, n_8, 32
- **q_{proj}**: Query weight matrix, W_0^1, dimensions $n_2 \times n_2$, typically 4096×4096
- **k_{proj}**: Key weight matrix, W_0^2, dimensions $n_2 \times n_2$, typically 4096×4096

- **v_{proj}**: Value weight matrix, W_0^3, dimensions $n_2 \times n_2$, typically 4096×4096
- **o_{proj}**: Output weight matrix, W_0^4, dimensions $n_2 \times n_2$, typically 4096×4096
- **gate_{proj}**: Gate weight matrix, W_0^5, dimensions $n_7 \times n_2$, typically 11008×4096
- **down_{proj}**: Down weight matrix, W_0^6, dimensions $n_2 \times n_7$, typically 4096×11008
- **up_{proj}**: Up weight matrix, W_0^7, dimensions $n_7 \times n_2$, typically 11008×4096
- **lm_{head}**: W_8, dimensions $n_1 \times n_2$, typically 32000×4096
- **mlp**: Feed-forward network comprising W_0^5, W_0^6, W_0^7
- **self_{attn}**: Self-attention mechanism comprising W_0^1, W_0^2, W_0^3, W_0^4

For Baichuan-7B, **W_{pack}** is a concatenated matrix three times larger, formed by column-wise concatenation of W_0^1, W_0^2, and W_0^3: $[W_0^1, W_0^2, W_0^3]$.

10 Parameter Count

Trainable parameters consist of W_j^1 through W_j^7 for $j = 0, 1, \ldots, n_8 - 1$, plus W_8. The total parameter count is:

$$n_8(4n_2^2 + 3n_2n_7) + n_1n_2 = 4n_2^2n_8 + 3n_2n_7n_8 + n_1n_2$$

For LLAMA-7B: 6,607,073,376 parameters.
For LLAMA-13B ($n_1 = 32000$, $n_2 = 5120$, $n_5 = 40$, $n_6 = 128$, $n_7 = 13824$, $n_8 = 40$): 12,851,609,600 parameters.
For Baichuan-7B ($n_1 = 64000$, other constants same as LLAMA-7B): 6,738,149,376 parameters.
For Baichuan-13B ($n_1 = 64000$, other constants same as LLAMA-13B): 13,015,449,600 parameters.

References

Jianlin Su. RoFormer: Transformer with Rotary Position Embeddings - ZhuiyiAI. Tech. rep. 2021. URL: https://github.com/ZhuiyiTechnology/roformer.

[2] Hugo Touvron et al. Llama 2: Open Foundation and Fine-Tuned Chat Models. 2023. arXiv: 2307.09288 [cs.CL].

[3] Hugo Touvron et al. "LLaMA: Open and Efficient Foundation Language Models". In: (2023). arXiv: 2302.13971 [cs.CL].

[4] 何沧平; 许涛; "BERT 模型的数学形式". In: (). DOI: 10.12074/202110.00071.

[5] 何沧平; 许涛; "大语言模型旋转位置编码的简易推导". In: (). DOI: 10.12074/202307.00071V2.