Details of the LLaMA Model

(**L**arge **L**anguage Model **M**eta **A**I)

By: Hiva Mohammadzadeh

Structure of the model

- A collection of foundation Encoder-Decoder language models ranging from 7B to 65B parameters.
- Works by taking a sequence of words as an input and predicts a next word to recursively generate text.
- The model shows that it is possible to train state-of-art models using publicly available datasets.
- The best performances are not achieved by the largest models but by smaller models trained on more and better data and trained for longer time.
- LLaMA is only trained on publicly available data, i.e., CommonCrawl, C4, GitHub, Wikipedia, etc, making it more compatible with open-sourcing.
- The goal is to make models that are cheaper at inference.

Training Scheme

- Pre-Training:
  a. Pre-trained on large, diverse text using an unsupervised learning approach. The model is trained to predict the next toke in the sequence give the past and present tokens
  b. They chose text from the 20 languages with the most speakers, focusing on those with Latin and Cyrillic alphabets.
- Fine-Tuning:
  a. Fine-Tune the model on specific downstream tasks by adding a task-specific output layer.

Structure Modifications

The authors propose changes to the transformer structure to make it more efficient which include:

- Add Pre-normalization (RMSNorm) to improve the training stability by normalizing the input of each transformer sub-layer instead of normalizing the output.
- Use SwiGLU activation function instead of ReLU or GeLU.
- Remove the absolute positional embeddings and add rotary positional embeddings at each layer of the network instead.
- Use AdamW optimizer.
- They also optimize the multi-head attention to reduce memory usage and computation.
- Reducing the number of activations that are recomputed during the backward pass by saving them
- Model and sequence parallelism to reduce memory consumption

RMSNorm Normalization Technique

- RMSNorm is an extension of layerNorm that computes the root mean square (RMS) value of the activations across all feature dimensions and channels, resulting in a single scalar value per example. This scalar value is used to normalize the activations, and a learnable scale parameter is applied to the normalized activations.
- More effective than LayerNorm in the presence of data with high variance. Gives the model re-scaling invariance property and implicit learning rate adaptation ability. Therefore, computationally simpler and more efficient.

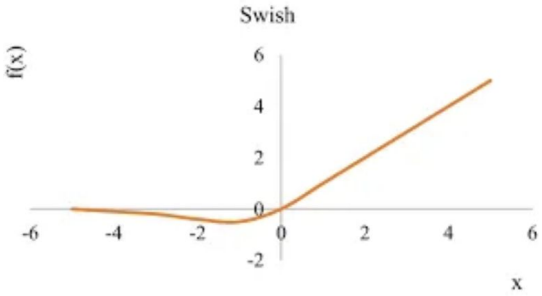
$$\bar{a}_i = \frac{a_i}{\mathbf{RMS}(\mathbf{a})} g_i, \quad \text{where } \mathbf{RMS}(\mathbf{a}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}.$$

SwiGLU Activation Function

- The SwiGLU (Swish-Gated Linear Unit) is a combination of the Swish activation function and the Gated Linear Unit (GLU).

$$\mathrm{FFN}_{\mathrm{Swish}}(x, W_1, W_2) = \mathrm{Swish}_1(xW_1)W_2$$

$$\mathrm{GLU}(x, W, V, b, c) = \sigma(xW + b) \otimes (xV + c)$$



- It is smoother than ReLU which allows it to have better performance and a faster convergence. It is also able to capture complex nonlinear relationships.

$$\mathrm{FFN}_{\mathrm{SwiGLU}}(x, W, V, W_2) = (\mathrm{Swish}_1(xW) \otimes xV)W_2$$

Rotary Embeddings (RopE)

- A type of position embedding which encodes absolute positional information with rotation matrix and naturally incorporates explicit relative position dependency in self-attention formulation.

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W^{(11)}_{\{q,k\}} & W^{(12)}_{\{q,k\}} \\ W^{(21)}_{\{q,k\}} & W^{(22)}_{\{q,k\}} \end{pmatrix} \begin{pmatrix} x^{(1)}_m \\ x^{(2)}_m \end{pmatrix}$$

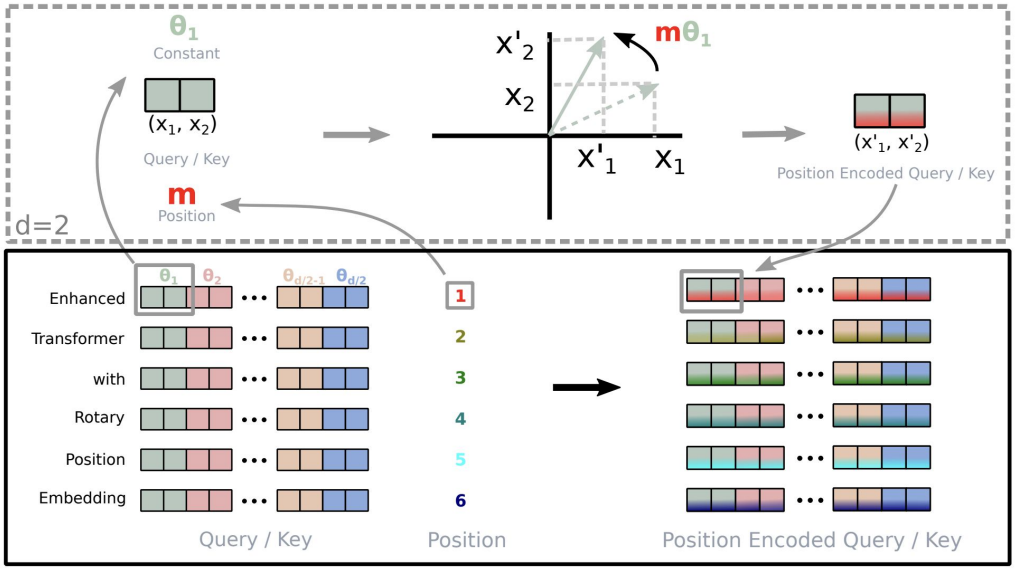
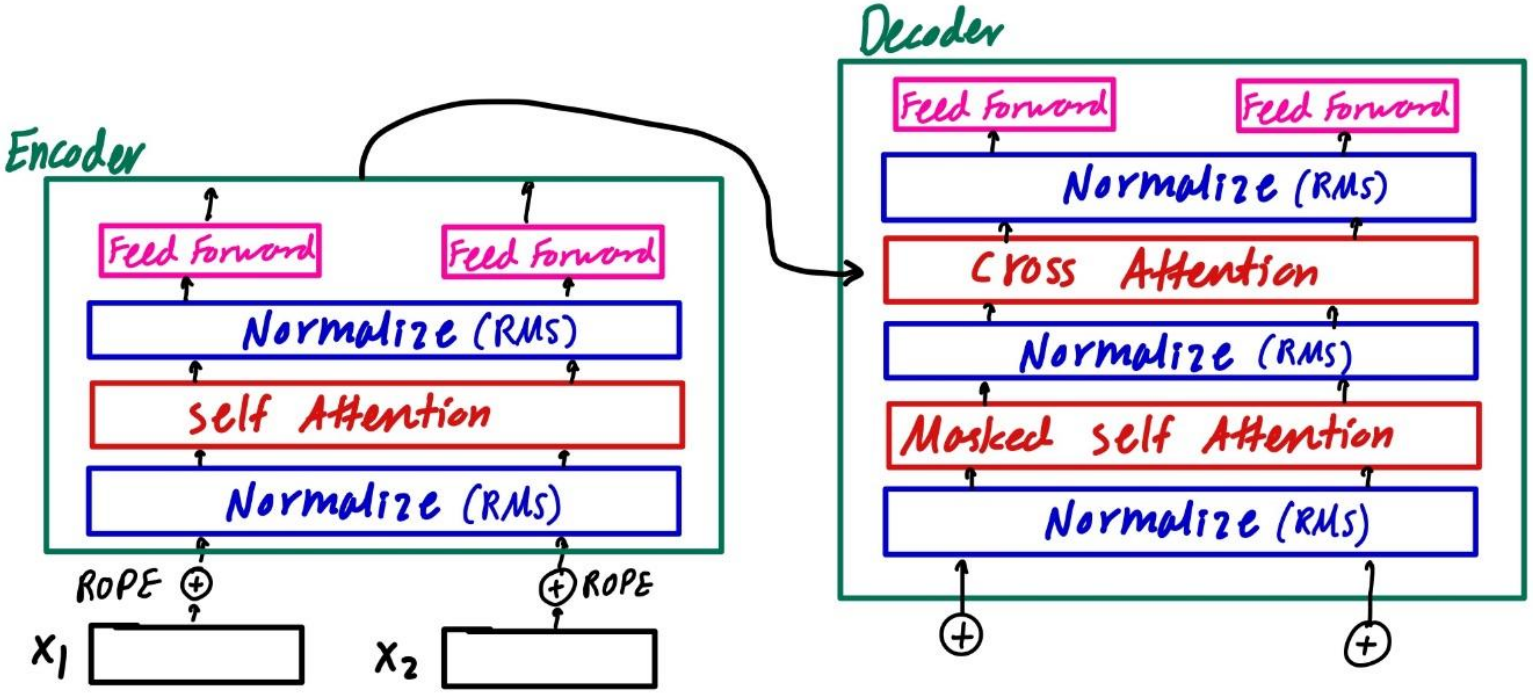


Figure 1: Implementation of Rotary Position Embedding(RoPE).

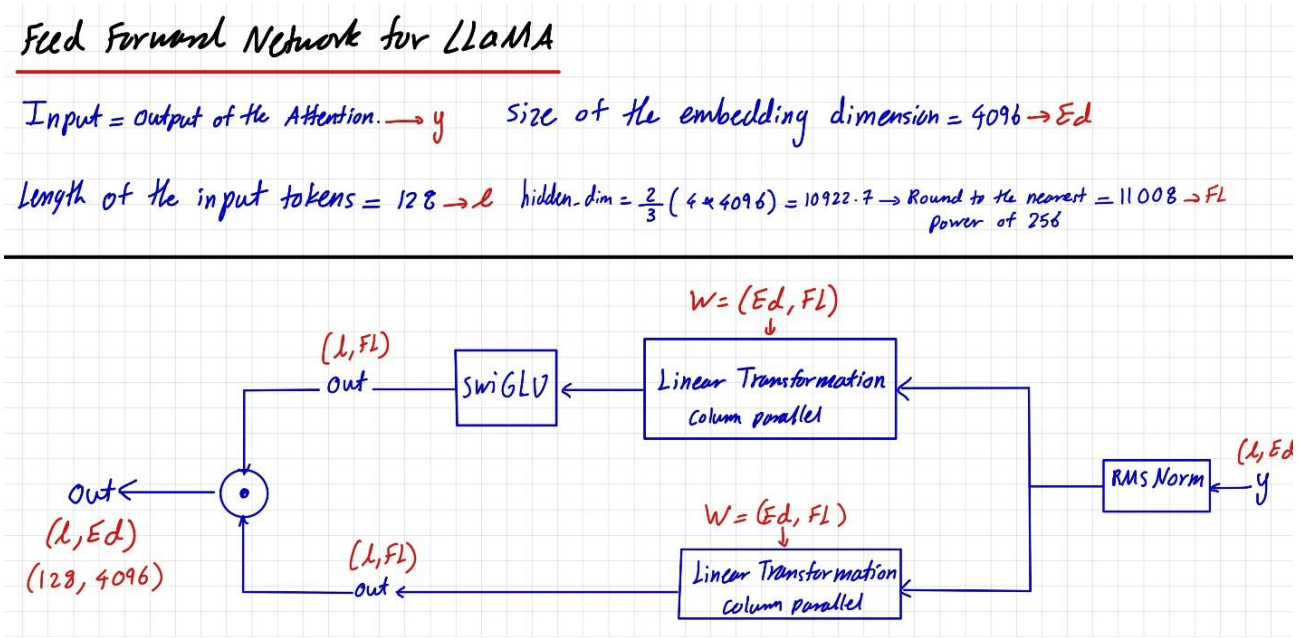
LLaMA Architecture

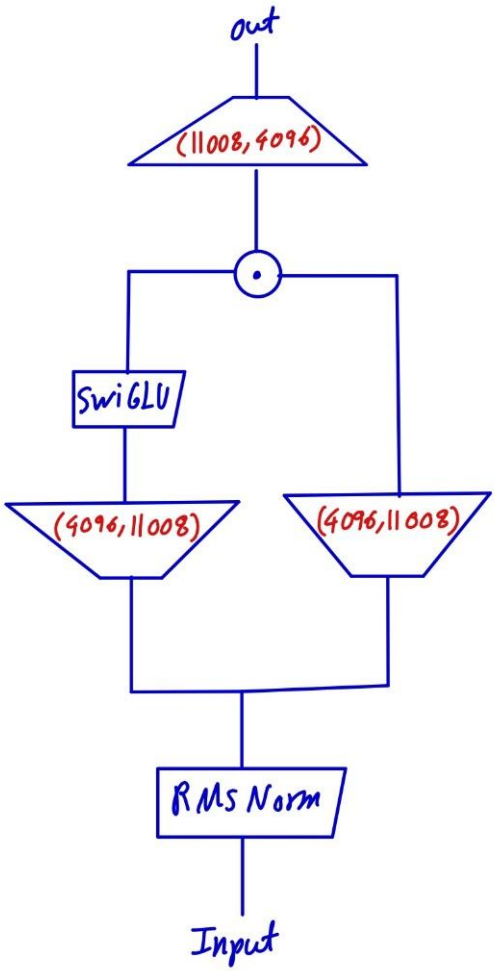
Transformer Encoder-Decoder from the “Attention is all you need” paper with some modifications:

- Feed Forward Network with the SwiGLU Activation function
- Normalize before every sub-layer



Feed Forward Network

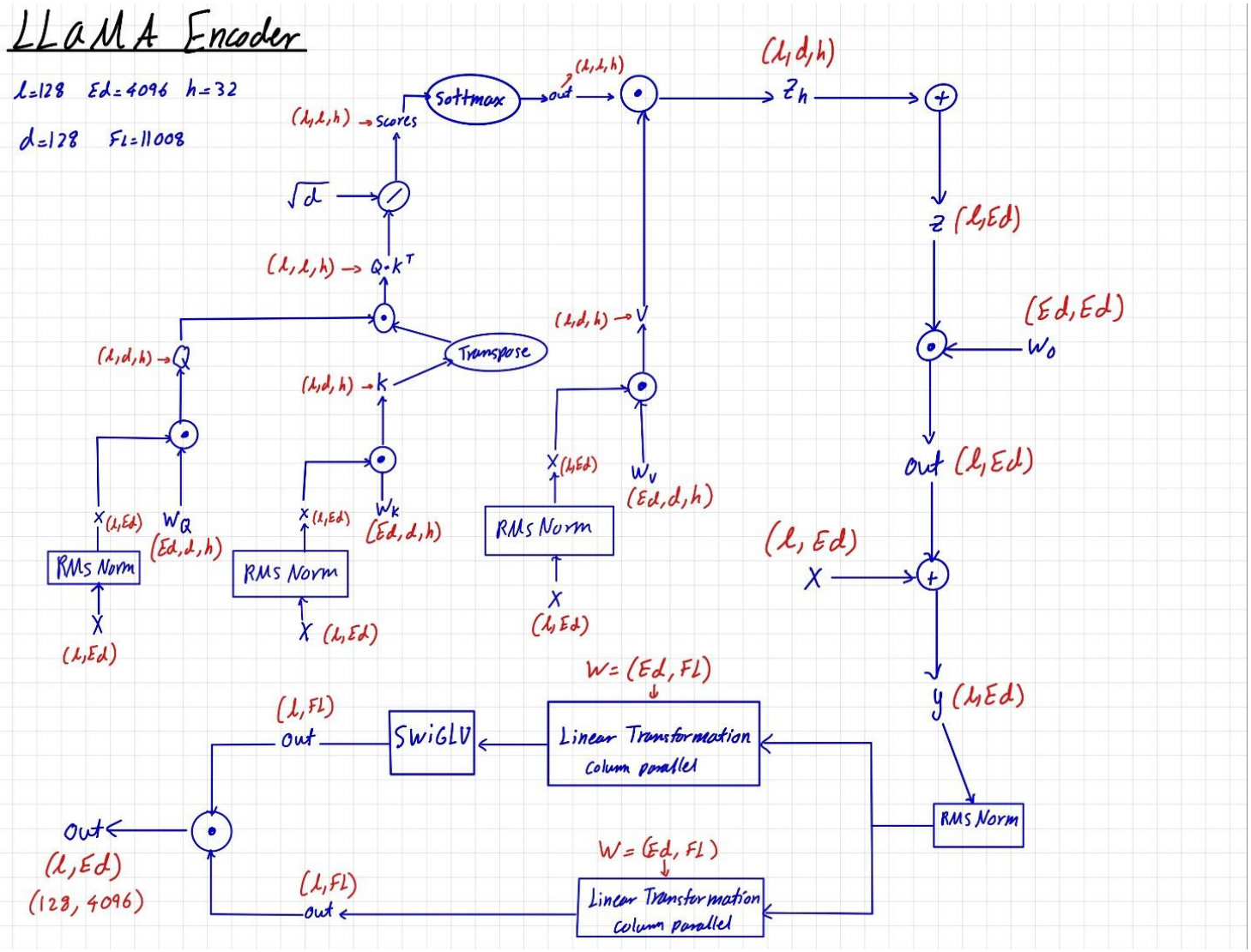




Dimensions of Weight Matrices

- The input embedding matrix has a dimension of (batch_size, sequence_length, embedding_size) = (batch_size, 128, 4096). But in order to do the attention parallel for different heads, we use (batch_size, 128, 4096/32 heads) = (batch_size, 128, 128, 32)
- The Wk, Wv, and Wq weight matrices used in the **self-attention mechanism** have dimensions of:
  - Query weight matrix: (batch_size, hidden_size, hidden_size) = (batch_size, 4096, 4096)
  - Key weight matrix: (batch_size, hidden_size, hidden_size) = (batch_size, 4096, 4096)
  - Value weight matrix: (batch_size, hidden_size, hidden_size) = (batch_size, 4096, 4096)
  - Output weight matrix: (batch_size, hidden_size, hidden_size) = (batch_size, 4096, 4096)
- The K, V, Q matrices also have dimensions of:
  - Query matrix: (batch_size, sequence_length, hidden_size) = (batch_size, 128, 4096) or (batch_size, 128, 128, 32)
  - Key matrix: (batch_size, sequence_length, hidden_size) = (batch_size, 128, 4096) or (batch_size, 128, 128, 32)
  - Value matrix: (batch_size, sequence_length, hidden_size) = (batch_size, 128, 4096) or (batch_size, 128, 128, 32)
- The weight matrices used in the **feed-forward neural network** have dimensions of:
  - First dense layer (column parallel): (4096, 11008).
  - Second dense layer (Row parallel): (11008, 4096).
  - Third dense layer on the input (Column parallel): (4096, 11008).

Computation Diagram of the Encoder



Computation Diagram of the Decoder