batch = B
seq_len = L
dim = D
n_heads = Nh
n_kv_heads = Nkv
hidden_dim = Dh
n_layer = Nl
vocab_size = V

Llama model architecture accoding to facebookresearch/ llama/ llama/ model.py and karpathy/ llama2.c/ model.py

A lookup table with 1 as input size, D as output size.
Input value is a unsigned int in the range of [0, V).
Output value is a vector of size D.

Run "Root Mean Square Normalization" on dimension D
`x * torch.rsqrt(x.pow(2).mean(- 1, keepdim= True) + epsilon)`
- rsqrt means reciprocal of the square- root
- x.mean(- 1, keepdim= True) outputs shape (B, L, 1)

Vector- by- matrix multiplication
- Vector is the dimension D of Xi'
- Matrix is learned weights, no bias

Split the last dimension D into multiple heads for Q
The same for KV, but their # of heads are smaller than Q

Qr = {Q0, Q2, Q4, ...}
Qi = {Q1, Q3, Q5, ...}

Kr = {K0, K2, K4, ...}
Ki = {K1, K3, K5, ...}

Constants shared by all layers
- m=[0,L)
  - position of token in the sequence
- #=10000.pow(- 2i/ d)
  - i=[0,d/ 2)
  - d= D/ Nh

Use memory space to trade for computational complexity

1. Broadcast constant matrix on dimension 0 and 2
2. Elem- wise 4D tensor multiplication

Foundamentally it is
1. Vector- by- matrix element wise multiplication
vector = Cos|Sin(1, ?, 1, D/ Nh/ 2)
matrix = Qx(?, ?, Nh, D/ Nh/ 2)
2. On top of 1, loop through L for both tensors
3. On top of 2, loop through B for Qx and keep Cos|Sin the same

Q = {Qr0, Qi0, Qr1, Qi1, Qr2, Qi2, ...}
K = {Kr0, Ki0, Kr1, Ki1, Kr2, Ki2, ...}

Duplicate the last dimension by Nh/ Nkv times
and fold it into 2nd to last dimension

Again, use memory space to trade for computational complexity

Move the Nh (# of head) dimension to batch domain,
so that scaled- dot- product is done
on L (sequence) and D (embedding) dimensions

Constants that are used to mask attentions
from current tokens to future tokens,
so that those attention value's softmax is 0,
by adding - # (negative infinity)

Calculate along the last dimension

$$y_i = e^{x_i} / \sum_i e^{x_i}$$

NOTE: V needs NO transpose

Back to (batch, sequence, embedding)

Dn (hidden dimension) is ~2.5x of D

Goes back to the input of Transformer Layer.
> Drawing the arrow will mess up PlantUML's engine
> and make this diagram unreadable, therefore it's not shown here.

From embedding to vocabulary probability. Each number
in the last domain represents the probability of corresponding token

Dh = 4 * 2 * D // 3 then round- up to multiple_of